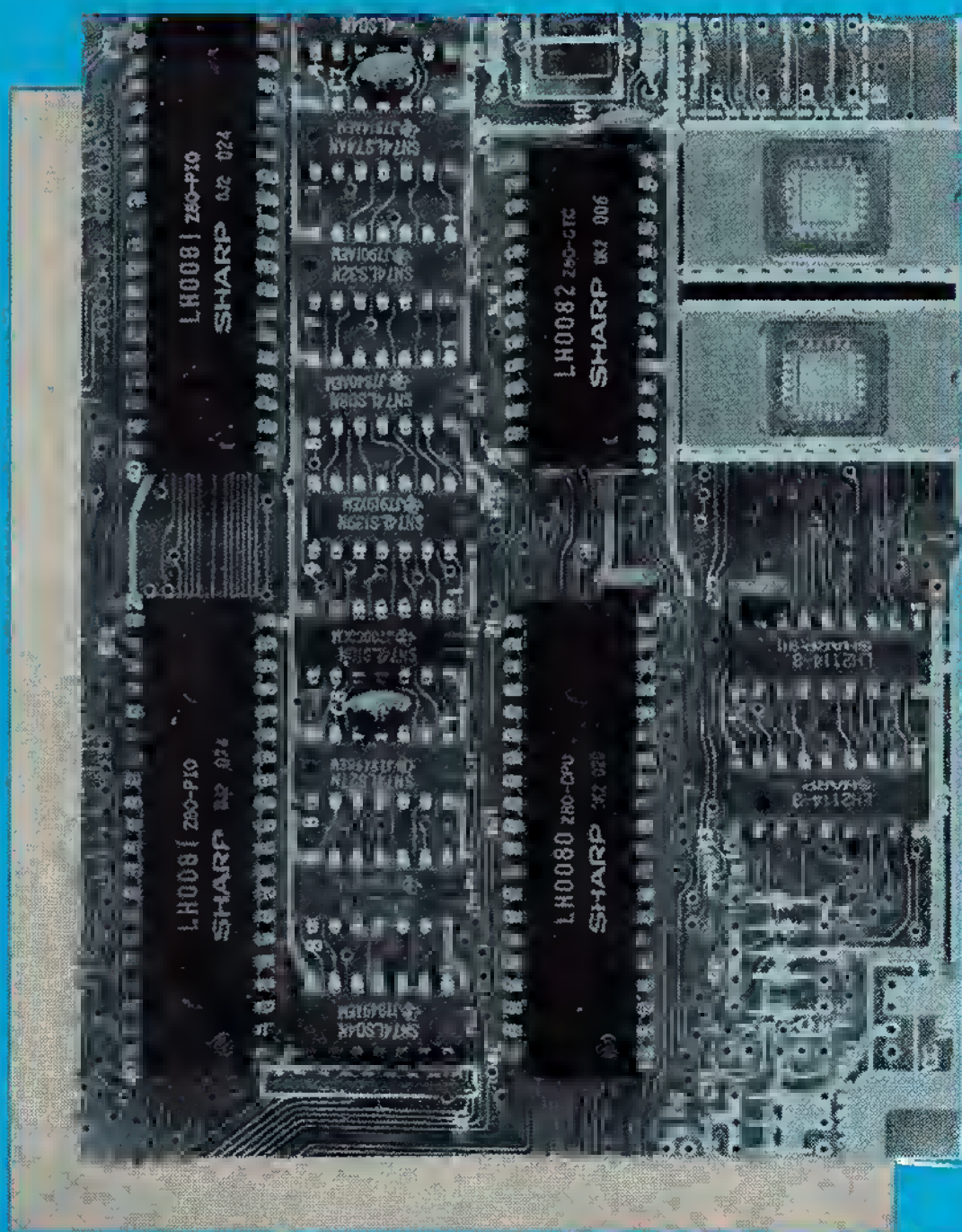


図解

マイクロコンピュータ

# Z-80の使い方

横田英一 著 ● オーム社



## 図解 初めて マイコンを学ぶ人のために

(A 5判 170頁)

鎌田信夫・野島晋 監修 大原茂之 著  
初めてマイコンを学ぼうとする人達を  
対象に、その機能と使い方の入口まで  
を、やさしく解説。

## 図解 マイコンの使い方

(A 5判 188頁)

小牧常松・大條廣 共著  
ハードウェア構成の概要と、ソフトウ  
ェアの構成の仕方、さらに近い将来一  
般技術者にも必要となるOSの概念に  
ついて、やさしく解説。

## 図解 マイコンのための BASIC入門

(A 5判 180頁)

小牧常松・大原茂之 共著  
実際にプログラミング技術を会得する  
ために、特にマイコン用プログラミン  
グ言語として目覚ましく普及している  
BASICを、例題を豊富に挿入し、わ  
かりやすく解説。

## 図解 マイコンのための アセンブラ入門

(A 5判 160頁)

大原茂之・倉田了一 共著  
最も一般的なインテル8080系に基づい  
て、アセンブリ語とアセンブラの使い  
方について、初学者にもわかるようや  
さしく解説。

## 図解 マイコンのための インタフェース入門

(A 5判 180頁)

大原茂之・北沢国男 共著  
インタフェースの基礎概念からアナロ  
グ量をマイコンに取込むまでの実際を  
図解により平易に解説。





**図解** マイクロコンピュータ

# **Z-80**の使い方

横田英一 著

オーム社

本書は、著作権法（法律第48号）第六条によって、著作権および出版権が保護されている著作物です。

☆ 複写複製する場合の御注意

本書の内容の一部あるいは全部を、無断で、複写機等いかなる方法によっても複写複製すると、著作権および出版権の侵害となる場合がありますので御注意ください。特に、学校・企業・団体等において、講習会、研修会、その他の目的のために複写複製する場合や、データベースとして利用されるためにコンピュータに入力する場合には、著作権者（著作者）および出版権者（オーム社）の許諾を得ないかぎり、著作権および出版権の侵害となります。

☆ 他書へ転載する場合の御注意

本書の内容の一部を他書へ転載する場合には、著作権者（著作者）および出版権者（オーム社）の許諾を得ないかぎり、著作権および出版権の侵害となります。

☆ 本書の複写複製、および内容の一部の転載等についてのお問合せは下記にお願いします。

〒101 東京都千代田区神田錦町3-1

株式会社 オーム社出版部（著作権担当）

Tel.03-233-0641

## すいせんの言葉

私達のまわりには、今大きな変革が起りつつあり、しかも猛スピードで、さらに加速されつつ事は進んでいます。

19世紀に起った第一次産業革命が、肉体労働の軽減を機械化で行ったように、現在の姿は頭脳労働の軽減や人間性の回復を電子技術の中核として実現されつつあります。特に2進法の数理から生れた4ビットのマイクロコンピュータは8ビットに飛躍し、さらに必要性により16ビット系に拡大しようとしております。

最も手近で最も多くの用途があり、汎用性の高い8ビット系がマイクロコンピュータの中心として、オフィスオートメーション、ホームコンピュータ、インテリジェンス機器等の主要電子部品の核であることは間違いないと信じます。

8ビット系マイクロコンピュータの中でも本書がご案内するZ-80は、ハードウェア的にもソフトウェア的にも、きわめて高度なマイクロコンピュータであり、アーキテクチャでも命令体系でもミニコンピュータに匹敵し、時にはしのぐほどになっています。

Z-80の命令体系はインテル社の8080Aの命令をすべて含み、容易に置換えられ、そのうえ8080にない多くの新しい命令やアドレス指定方式が加えられています。

たとえば文字列の検索は準備操作によって1命令で実行でき、これは他のものの4命令と等価になっています。

さらにZ-80には、その能力を補完する周辺デバイスとして、PIO, SIO, CTC, DMA等のファミリーが用意されています。

世間には数多くの良書が出ていますが、本書はわかりやすい手ほどきのものとして、ご愛読、ご再読をおすすめします。

1981年4月

シャープ株式会社

取締役 経澤 崇泰





## は し が き

マイクロコンピュータがエレクトロニクス分野に画期的な進歩をもたらしたのも、早くも過去の語り草になろうとしております、すでにマイクロコンピュータは多くの応用製品を生み出し、研究熱心なユーザによって秘められた能力をさらに深く追求されて、あらゆる分野に恩恵をもたらしております、

今後の電子技術の一翼としてマイクロコンピュータ応用技術が重要な位置を占めるであろうことは自明です、学校や企業の実験室には、テスタやシンクロスコープと並んでプログラム開発用の装置が導入され、資料室には回路図といっしょにプログラムが保管されることになります、

単なる時流やはやりものではない、根底的な変遷の中にあって、マイクロコンピュータ応用技術の取得は、真空管がトランジスタにおき変った以上に早いスピードで必要性を増しております、

マイクロコンピュータは、1971年にインテル社から発表されたI4004以来、10年間の間に数回の世代交替がありました、この間に機能は飛躍的に拡大し、処理速度、プログラムサイズ、部品構成、応用技術等の面で数段の進歩をとげてきました、8ビット系では、そのほぼ頂点にあるのがザイログ社のZ-80ファミリです、8ビット系でこれ以上の機能を要求すれば16ビット系へ移行するのが一般的な考え方ですが、最も多くの用途があり汎用性の高い8ビット系は今後もマイクロコンピュータの中心にあることに変わりないはずです、

本書では、エレクトロニクス技術の研さんに励む学生諸氏、および企業にあってマイクロコンピュータ導入を目前に技術修得を課題とする電子技術者諸氏のために、今後も主流の地位を他に譲ることのないであろうZ-80を例に、動作原理から応用例までを解説したものです、本書でZ-80をマスタしてしまえば、どんなコンピュータでも容易に理解できると信じます、

マイクロコンピュータは、ハード面、ソフト面が有機的に結合し合いシステム

を形成していますので、ページに従った展開による一次元的な説明では到底表現しきれないものです。平面的、二次元的なつながりを重要視する意味で項目間の参照は不可欠です。また、一読して理解できないことを嘆くなら再読を勧めます。1回目ではぼんやりと全体像をとらえ、2回目では後を書いてあることを思い起こしながら前の説明を見ていきます。次に自分でプログラムを考えてみてください。例題のプログラムを自分なりに手なおしするだけでもよいのです。自分自身で試行することにより不明点をとらえ、解説を読みなおせば確実に身につきます。マイクロコンピュータは物性や自然科学とは違って、人間が考え出して作ったものです。使いやすく、覚えやすくするためにさまざまな知恵をしばってあります。自然現象の探究とは異なった考え方で対処すれば、いたって気軽に入っていけるものと思います。

本書では、基礎的な項目について詳細な解説は、他に良書が多数あることを理由に、ふれる程度にとどめた点もあります。オームの法則に始まる電子回路、ブール代数、論理素子(TTL)についての最低限の知識は必要です。また、マニュアルにある正確緻密な表現よりも、平易な解説を心がけました。用語がわからないだけでコンピュータは理解できない難物とってしまうことも多く、この点も読み進むうちにだんだんとなじめるように配慮したつもりです。

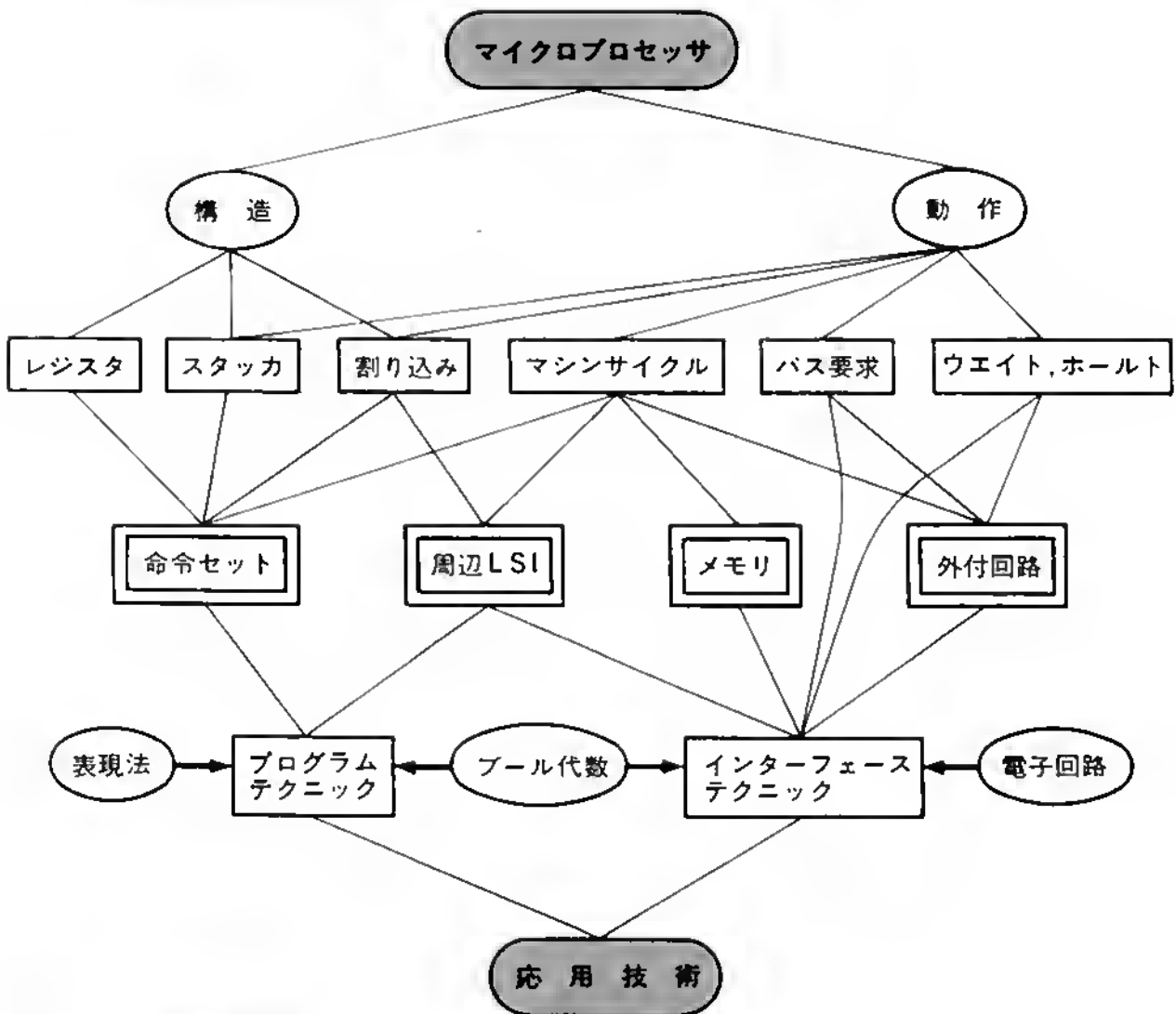
変貌する電子技術の中であって、新しい分野への進出を目ざす読者諸兄の勉学の一助となり、エレクトロニクス産業界の発展のためのごく小さな後押しになれば筆者の意とするところであります。

最後に、執筆にあたり御理解と手間をいとわない御協力を賜ったシャープ株式会社の関係部課の皆様、オーム社出版部の方々に心から感謝の意をささげる次第であります。

1981年4月

著者しるす

# マイクロプロセッサの応用技術



## 基礎知識

- |                                       |                                 |                                       |
|---------------------------------------|---------------------------------|---------------------------------------|
| ◦ 表現法<br>(用語)<br>(16進表現)<br>(アセンブリ言語) | ◦ ブール代数<br>(真理値表)<br>(TTL ロジック) | ◦ 電子回路<br>(オームの法則)<br>(電子部品)<br>(半導体) |
|---------------------------------------|---------------------------------|---------------------------------------|

## 付加事項

- 経験と知識に基づく 慣れ≒ヤマカン を養うこと。  
自分でやってみることが重要
- 中身がどうなっているかは問題ではない。  
どうすればどうなるかを知って使いこなすことが重要。



# 目 次

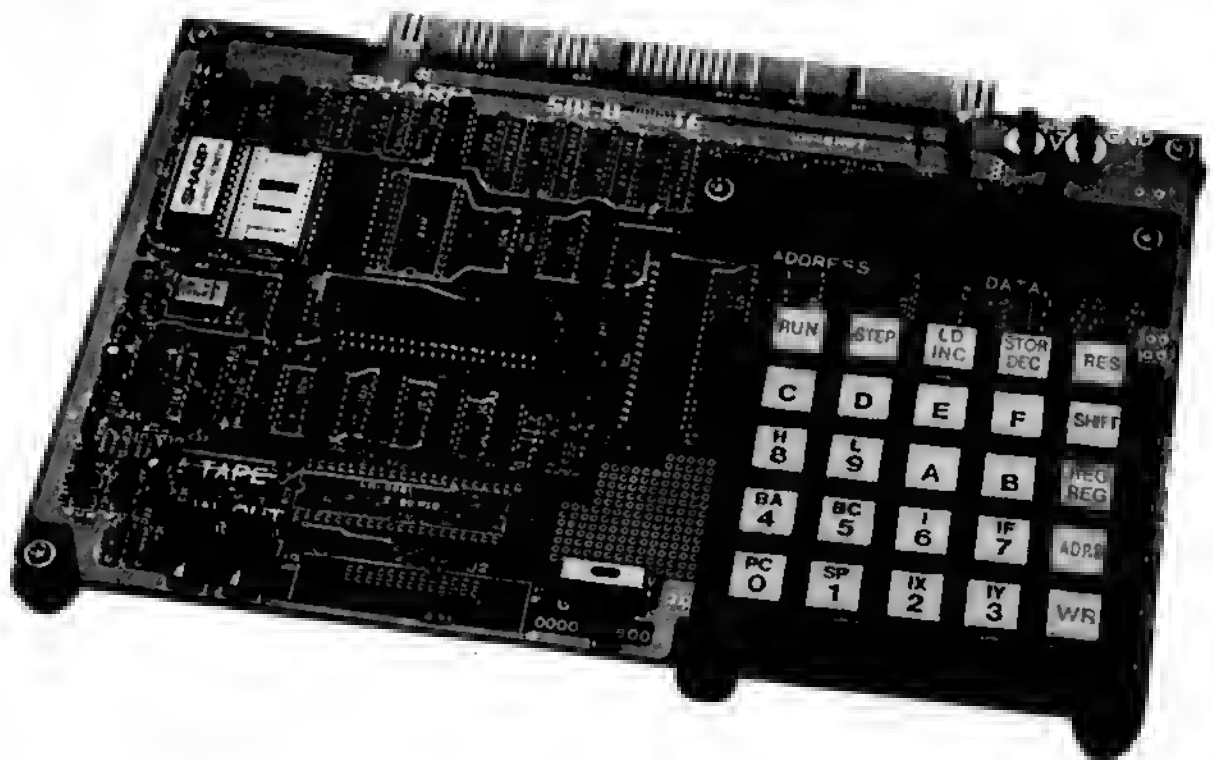
1	マイクロプロセッサの特質	2
2	マイコンシステムの種類	4
3	特質を活かすシステム設計	6
4	Z-80 ファミリの特徴	8
5	Z-80 CPU	10
6	Z-80 PIO	12
7	Z-80 CTC	14
8	Z-80 DMA	16
9	Z-80 SIO	18
10	メモリの種類と用途	20
11	ビットパターンと16進表現	22
12	プログラムの実行	24
13	CPUの信号のやりとり	26
14	データバス、アドレスバスとシステム制御信号	28
15	命令語の構成	30
16	命令の実行	32
17	アセンブラ記法のルール - 1 -	34
18	アセンブラ記法のルール - 2 -	36
19	命令の分類	38
20	メモリ空間とIO空間	40
21	アドレスデコーダ	42
22	バスバッファ	44
23	クロック	46
24	リセット	48

25	システム構成	50
26	フェッチサイクルの動作	52
27	メモリリードサイクル	54
28	メモリライトサイクル	56
29	IO リードサイクル, IO ライトサイクル	58
30	リフレッシュサイクル	60
31	CPU と周辺の接続 (インターフェース)	62
32	ウェイト信号とホルト信号	64
33	割り込みの概念	66
34	ノンマスカブルインタラプト ( $\overline{\text{NMI}}$ )	68
35	インタラプト ( $\overline{\text{INT}}$ )	70
36	モード 0 のインタラプト	72
37	モード 1 のインタラプト	74
38	モード 2 のインタラプト	76
39	デージェチェーン	78
40	バス要求と応答	80
41	内部レジスタの構成	82
42	A, I, R, F レジスタ	84
43	汎用レジスタ	86
44	補助レジスタと交換命令	88
45	IX, IY レジスタ	90
46	スタッカとスタックポインタ (SP レジスタ)	92
47	転送命令	94
48	算術演算命令	96
49	論理演算命令	98
50	ビット操作命令	100
51	ローテート, シフト命令	102
52	ブロック転送, ブロックサーチ, ブロック入出力命令	106
53	ジャンプ命令	110

54	コール、リスタート、リターン命令（サブルーチン）	114
55	Fレジスタとフラグ変化	116
56	2進化10進数と10進補正命令	118
57	ペリフェラルのプログラミング	120
58	PIO モード 0 の動作	122
59	PIO モード 1 の動作	124
60	PIO モード 2 の動作	126
61	PIO モード 3 の動作	128
62	PIO のプログラミング	130
63	PIO のプログラム例	132
64	CTC カウンタモード	134
65	CTC タイマモード	136
66	CTC のプログラミング	138







ワンボードコンピュータ製品例 SM-B-80TE(シャープ)

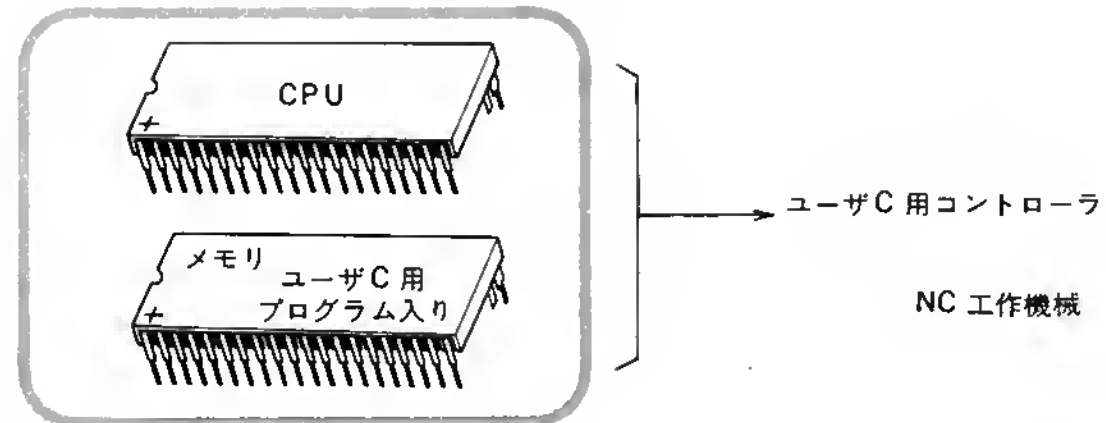
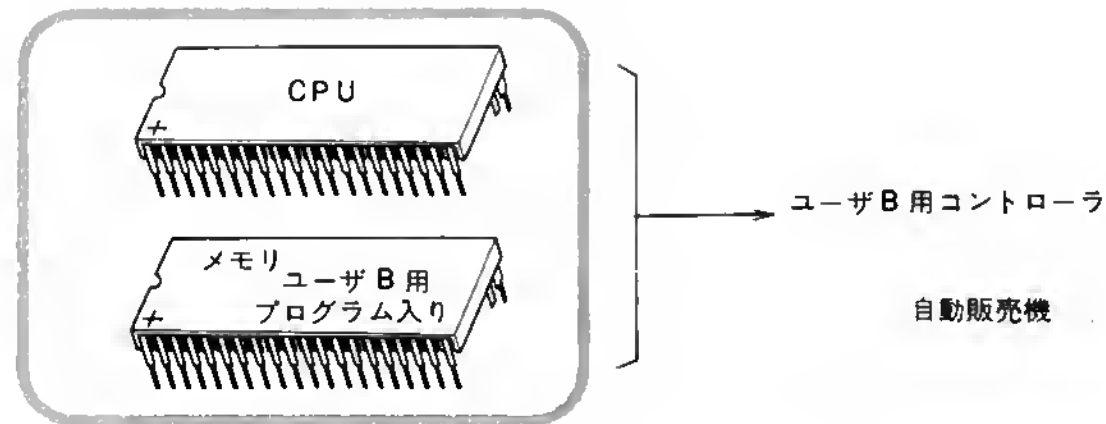
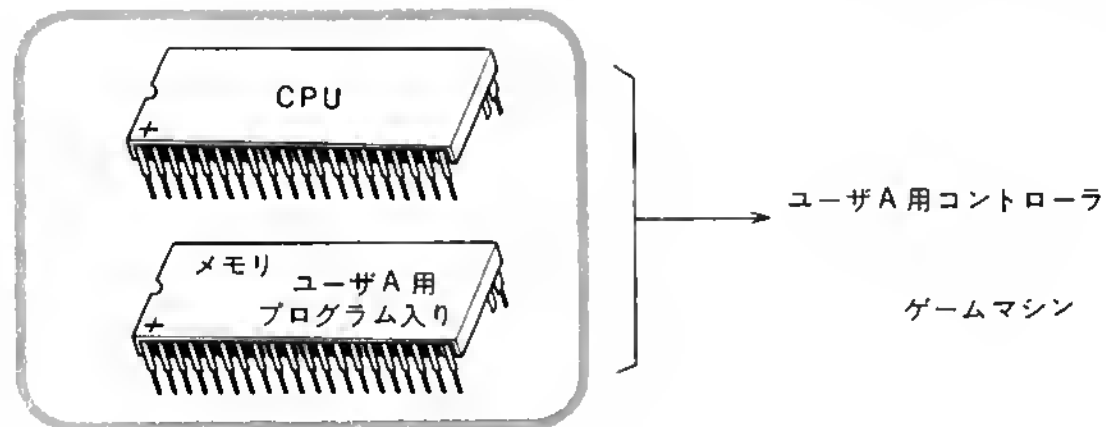
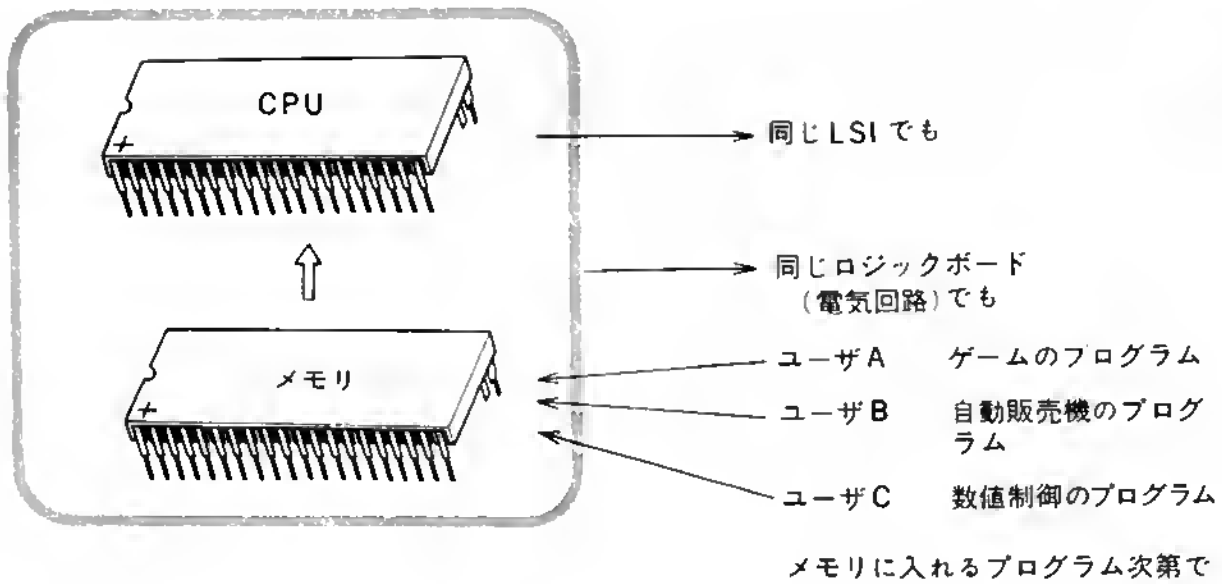
## ■ マイクロプロセッサの特質

コンピュータという機械が発明された目的は、人間には不可能な複雑な計算を短時間に行なうことでした。たとえば円周率の計算を一生の大半を費やして行なっても数百桁だったのに、コンピュータなら瞬時にもっと多くの桁を間違いなくやってのけることができます。人間を月へ送り込むのも、正確な軌道計算が刻々と得られるためにできたことで、コンピュータなしでは考えられないのです。ある手順の決まった仕事を、高速かつ正確に処理することのできる機械、これがコンピュータの目的であり、その目的は十分に達せられたといえるでしょう。

ところが、ストアード プログラム方式すなわち、処理手順を示すプログラムを、コンピュータに入力すべき情報の一つとして扱うことのできる今日のコンピュータ方式では、当初の目的以外に、大きな特徴が認識されました。全く同一の機械（ハードウェア）に対し、使用者が作成するプログラム（ソフトウェア；これは使用者によってそれぞれに異なる）をメモリに入れることにより、使用者のそれぞれの目的に合った仕事をする、という特徴です。コンピュータ メーカーは同じ機械を大量に生産すれば、科学計算であろうと、在庫管理であろうと、人事管理、生産ラインコントロール、座席予約、相性判断まで、プログラム次第で適用されてしまうのです。

半導体の技術が進歩して数千、数万の部品を数ミリ四方の中へ作り込めるようになったとき、専用の機能を持った LSI が、さまざまな目的に合わせて作られました。ところが大量生産にしか向かない LSI の弱点を補うべく、半導体メーカーは汎用 LSI の思想をコンピュータに見出したのです。コンピュータを LSI 化すれば、多く用途のある、つまり大量に売れる LSI が作れると考えたのです。マイクロプロセッサはこのような中から誕生し、プログラムによって機能が決定される汎用の論理素子として、すばらしい発展をとげ、現在も発展しつつある、“電子部品”なのです。マイクロプロセッサを中心としたマイコン システムを利用しようとする場合、この辺を的確に把握してかかることが重要です。

# 1 マイクロプロセッサの特質



## 2 マイコンシステムの種類

マイコンと呼ばれるコンピュータ システムには、いくつか考えられます。一つに、パーソナル コンピュータやオフィス コンピュータに近い性能を持つものを“**マイコン**”と呼びます。これらは、コンピュータを小さく、安価にまとめ、事務所や大学の研究室や個人の知的玩具としてまで普及した個人用コンピュータ システムです。電気の知識がなくても、簡単な言葉を覚えれば誰でも使えるテレビ、ステレオなどと肩を並べる全く新しいマニアライクな道具といえます。この場合のマイコンは、“**My Computer**”と解釈すべきでしょう。

もう一つの“マイコン”は、パソコンほどのはなばなしはありませんが、工作機械や自動販売機や家電製品に組み込まれている、機器制御用の**ワンボードマイコン**です。基板上に作られたマイクロ コンピュータ システムは、産業用としての中心をなすものです。各種の用途でそれぞれの目的に合った設計がなされ、特徴を出しています。組み込まれた時点では、プログラムは固定化され、機器のユーザには、コンピュータとしての使用はできないのが普通です。

1 個の LSI 上にメモリや入出力ポートを作り込んだ**ワンチップ マイコン**は、時計や電卓に使われています。中身を分析すれば、マイクロ コンピュータに違いありませんが、LSI を外面から見た場合、一つの機能を持った専用 LSI になってしまいます。開発過程では、コンピュータの特徴が活きて、短時間に安価な開発経費で専用 LSI が作れるのです。機器制御用マイコン システムをワンチップ化したと考えればよく、生産数量がきわめて大きい場合、コストやサイズ、消費電力などの点で、効果を上げることができます。

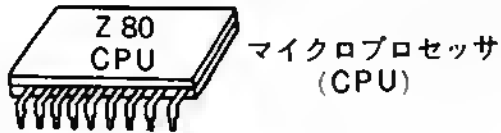
また、マイコン システムを構成する最重要部品であるマイクロ プロセッサ、つまり CPU の LSI を“マイコン”と呼びます。しかし、これだけではコンピュータとはいえず、周辺やプログラムを含めてコンピュータ システムが完成するのですから、あくまで CPU と呼ぶべきです。

## 2 マイコンシステムの種類

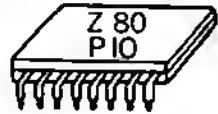
ワンボード・マイコン  
やパーソナル・コンピ  
ュータはマイコンシス  
テムの応用製品

パーソナル・コンピュータ

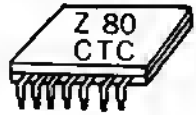
一方、ワンチップマイ  
コンは開発過程がコン  
ピュータである、専用  
LSI である



マイクロプロセッサ  
(CPU)



入出力 LSI

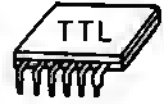


カウンタタイマ  
など

周辺  
LSI



メモリ



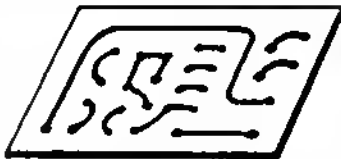
TTL IC



抵抗

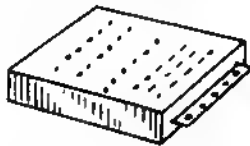


コンデンサなど



基板  
コネクタ  
など

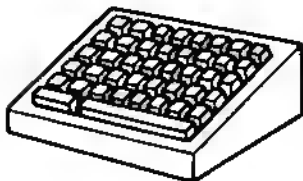
ワンボード  
マイコン  
(制御用)



電源



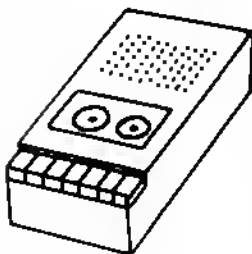
ブラウン管  
プリンタ  
など



キーボード



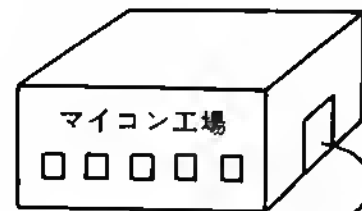
ソフトウェア



カセットテー  
プレコーダ  
フロッピー装置  
など



プログラム

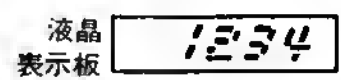


マイコン工場



電卓用 LSI

電卓工場



液晶  
表示板

電卓用  
LSI



電池

キーボード

1	2	3	C
4	5	6	/
7	8	9	%
+	-	×	÷

## 3 特質を活かすシステム設計

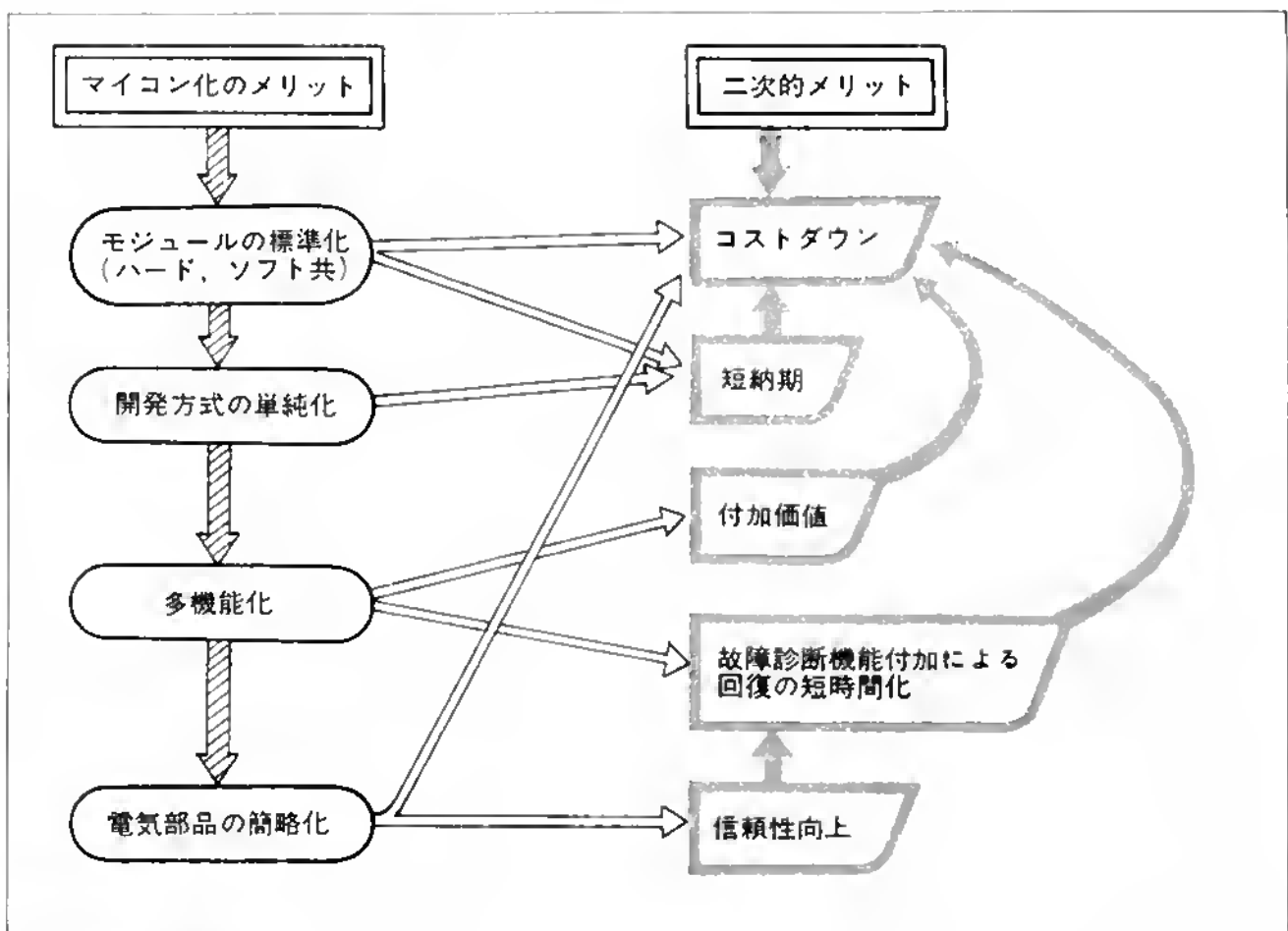
コンピュータは、仕事を高速に処理することと、同じ機械（ハードウェア）でもプログラム（ソフトウェア）を変えるだけで異なった働きをさせられること、の2点が最大の特徴です。制御用マイコンは、メカニズムやリレーや論理回路によるワイアードロジックからの延長線上にあり、高速性では、TTLロジックには一歩ゆずります。むしろ、プログラムにかかる機能面の柔軟性が大きなメリットです。また、多機能化することが比較的容易であり、故障時に自己診断をさせることすら可能です。マイコンボードとして決定された能力の限界以内であれば、同一の規格化された基板を何種類もの製品に応用することができ、開発コストの引き下げに効果が期待できるようになります。プログラムもモジュール化し保管をすることにより、経験の蓄積ができ開発期間の短縮がはかれます。

マイコンシステムの設計にあたっては、ハードウェアとソフトウェアの分担を決定することがポイントになります。回路設計とプログラム設計の前段階としてのシステム設計といわれる作業です。往々にして、電子技術者が設計したシステムはハードにウェイトがおかれてしまう傾向があります。ハード、ソフト共に精通した人が、目的の機能と、開発コスト、ランニングコスト、メンテナンスコストなどすべてにわたって検討して決定すべきです。特に部品やボード、プログラムの標準化という意味から、将来にわたって要求される機能の変動を見通すことが重要になります。変動要素はソフトウェアの分担とし、メモリの交換だけで対応できなければなりません。

概してコンピュータの特質を考えるならば、タイミング的に不可欠な部分をハードウェア化し、他はソフトウェアで実現する方法に利があるといえます。ハードウェアを最小限に止めることは、材料費、経年劣化、故障率を低減する上で有利であることは明白です。

### マイコン適用分野

- 多機能であって、ICレベルの論理素子で構成すると、大規模になりすぎるもの  
——→ パーソナルコンピュータ
- 大同小異の機種が多く、いちいち論理素子で構成したのでは設計に手間がかかりすぎるもの ——→ 端末装置、NC 機器
- 納入先ごとの変更があるもの ——→ ビル防災システム、ホテル管理システム
- あとで変更のあり得るもの ——→ 自動販売機、料金計算機
- 開発時間の限られる場合 ——→ ハードウェアは標準品を使い、プログラムだけを入れ換える



### 必要条件

- 改善意欲 ——→ 社内規格の統一・コストダウン
- 先行投資 ——→ 開発装置購入
- 技術力 (技術吸収力) ——→ 要員教育

## 1 Z-80 ファミリの特徴

これまでに最も普及したマイクロ プロセッサにインテル社の 8080 系があります。日本でも各社がセカンド ソースを発表しています。

8080 系は、8008 → 8080 → 8080A → 8085 と進化したのですが、この流れの延長線上に全く新しい考え方をとり入れて設計されたのが、ザイログ社の Z-80 です。したがって 8080 系の機能はすべて引き継いでおります。

Z-80 は、ハード的にはできるだけ少ない部品で構成できるようになっています。ファミリは 5 種類しかありませんが、これらの LSI に機能を分散させてあります。ファミリを構成する LSI 間のインター フェースは、多少複雑ですが、Z-80 の設計者の意図からはみ出さない限り問題になることはありません。むしろ、部品点数の削減に大きく貢献しています。クロックは単相の方形波でよく、電源は 5 ボルト単一で働きます。

8080 系の持つ命令は、すべて Z-80 の命令群の中に含まれていて、マシン語も同じです。アセンブリ語は、かなり差異がありますが、整理統合されており、大変覚えやすい言語になっています。命令数はきわめて多く、プログラム容量の短縮と、処理速度の高速化に大いに役立っており、わかりやすい言語体系とあいまって、開発期間の短縮が期待できます。

割り込みの機能で、Z-80 に新たに追加されたモードは、メモリ アドレスを自由に使ってプログラムが組めるようになっています。また周辺の LSI が何個つながっても（限度はありますが）、標準的な接続で割り込み処理ができます。特に制御用の応用面では重宝です。

総合的に、Z-80 は 8 ビットのプロセッサとしては、いまのところ最上位の機能を持ち、将来もこれに代わるものは必要とされないだろうとさえいわれています。もっと高機能が望まれるときは、CPU を複数にして分散処理をするか、16 ビット以上の機種を使用するほうが、8 ビット・プロセッサにさらに高度な機能を要求するより自然な流れではないでしょうか。



**Z-80ファミリ****Z-80 CPU (Central Processing Unit)**

Z-80 PIO (Parallel Input/Output Interface Controller)

Z-80 CTC (Counter Timer Circuit)

Z-80 DMA (Direct Memory Access)

Z-80 SIO (Serial Input/Output Interface Controller)

**特 徴**

- +5V 単一電源 → TTL コンパチブル
- 単相クロック → クロックジェネレータ不要
- 割り込み機能 → コントローラ不要、プログラムが簡単
- レジスタ群 → プログラム容量の圧縮・スピードアップ
- 命令セット → プログラム容量の圧縮・スピードアップ
- リフレッシュ機能 → ダイナミックメモリ使用可能 → コストダウン

**命令セットの特徴**

- 16ビットの引き算
- レジスタ、メモリのビットのセット、リセットテスト
- メモリブロックの転送、サーチ、入出力
- 2の補数をとる命令
- 4ビット単位のローテーション
- 整理されたモニタック（暗記用命令コード）

## 5 Z-80 CPU

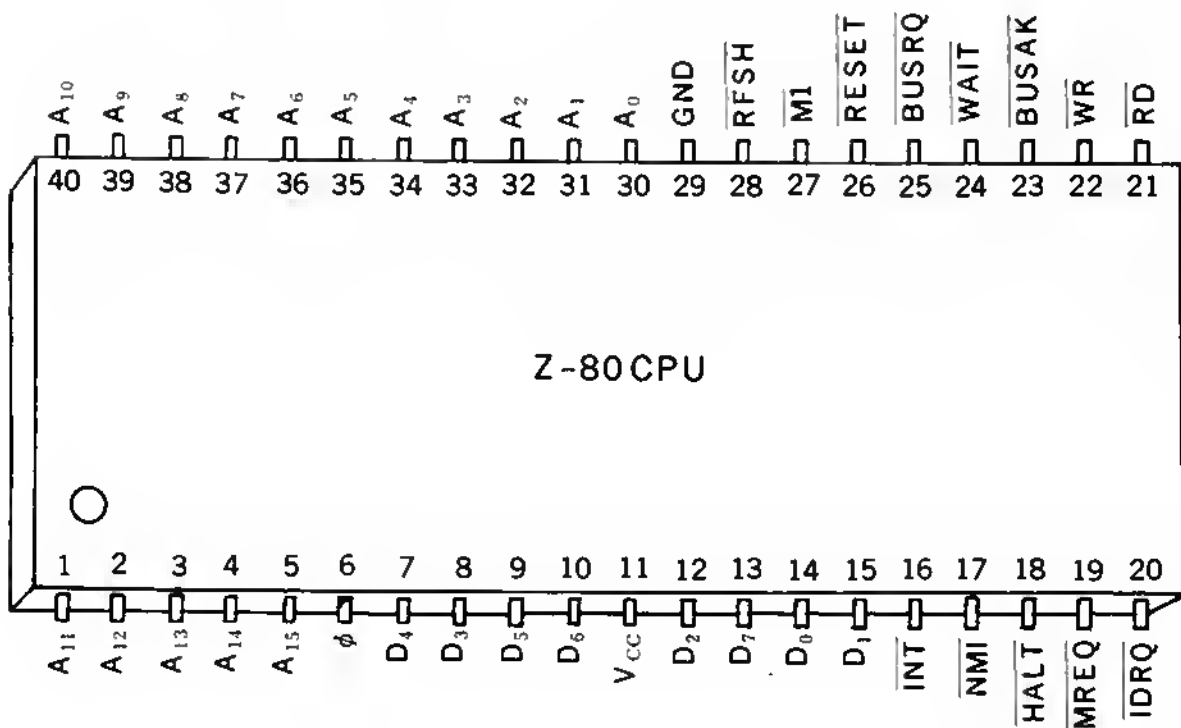
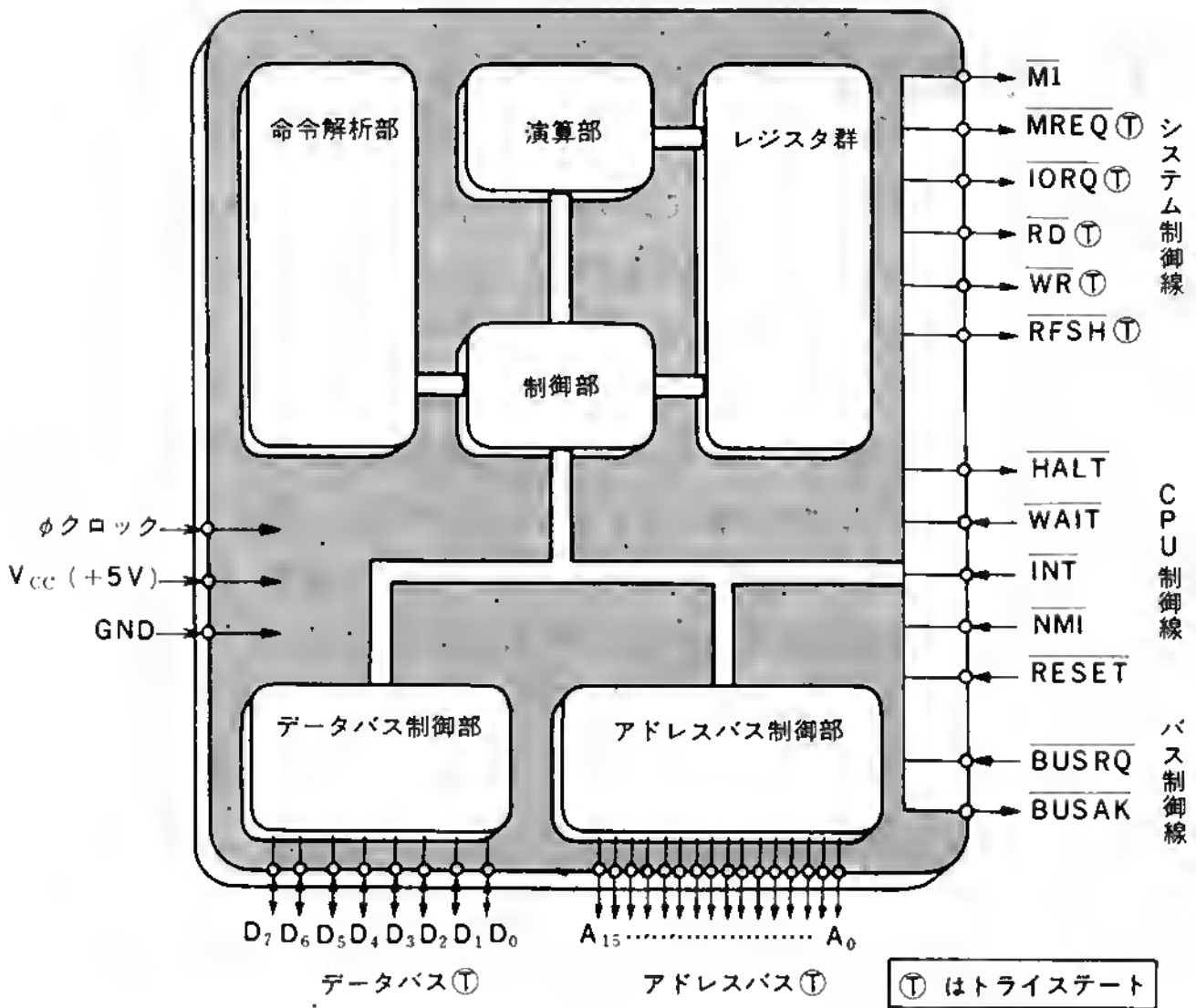
**CPU (Central Processing Unit)** は Z-80 ファミリの中核をなすものであり、ファミリの性格を決定する中心要素です。なお、ファミリで CPU 以外の LSI を、周辺 LSI とかペリフェラルと呼ぶことがあります。

Z-80 CPU には 8 ビットの汎用レジスタが 12 あります。組み合わせて 16 ビット レジスタとして使用することもできます。インデックス レジスタは 2 個あり、他にスタック・ポインタ、プログラム カウンタ、割込ベクトル レジスタ、メモリ リフレッシュ レジスタ、フラグ レジスタ、アキュムレータがあります。

これらのレジスタや命令を解析し実行する制御部や演算部の働きは、以下の項で解説します。

外部端子には、アドレス バス 16 本、データ バス 8 本、システム制御線 6 本、CPU 制御線 5 本、バス制御線 2 本、それにクロックと電源が出ています。図で、トライ ステートと記入されている端子は、“H” と “L” の二値状態のほかに、端子と内部が電氣的に切り離された“ハイ インピーダンス”状態を持つものです。動作と関係のないときはこの状態になっており、外部からの負荷にならないようになっています。

クロックは単相でよく、通常 2.5 MHz 以下、A バージョンは 4 MHz 以下で働きます。通常のバージョンと A バージョンはクロックの最大値が異なりますが、他の機能や電氣的特性は共通です。以下のファミリにもすべて通常のものとは A バージョンがあります。同じバージョンの LSI を揃えて使わなければ意味がありませんが、A バージョンの LSI を 2.5 MHz 以下で使用しても問題はありません。



## 6 Z-80 PIO

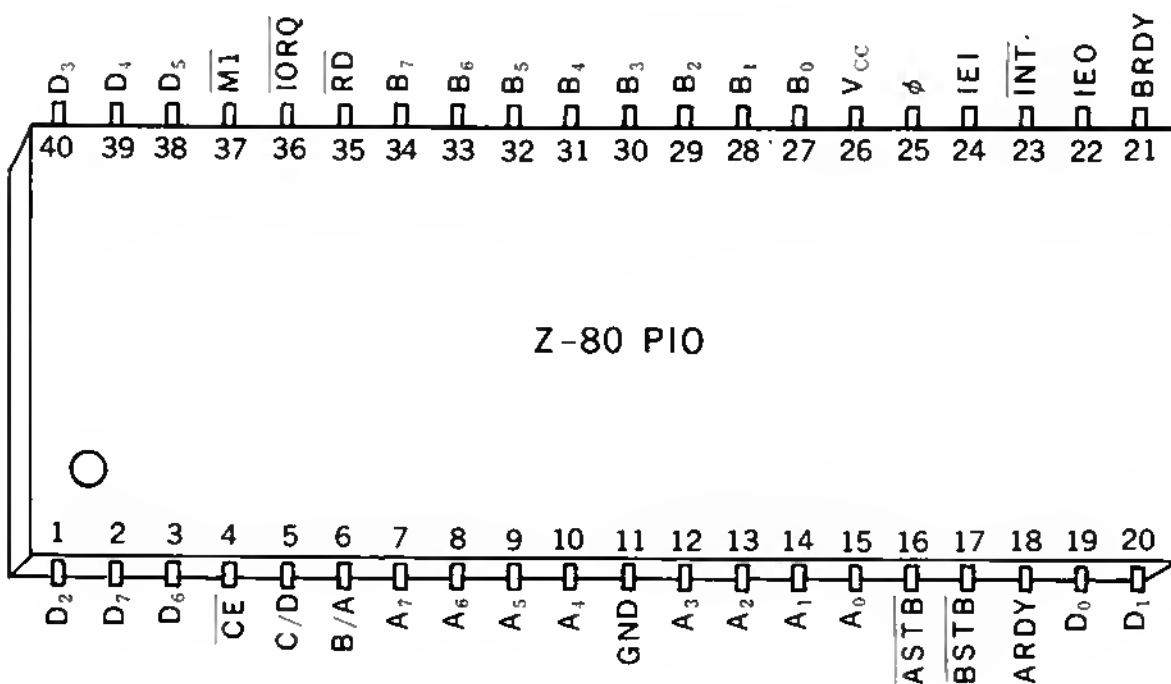
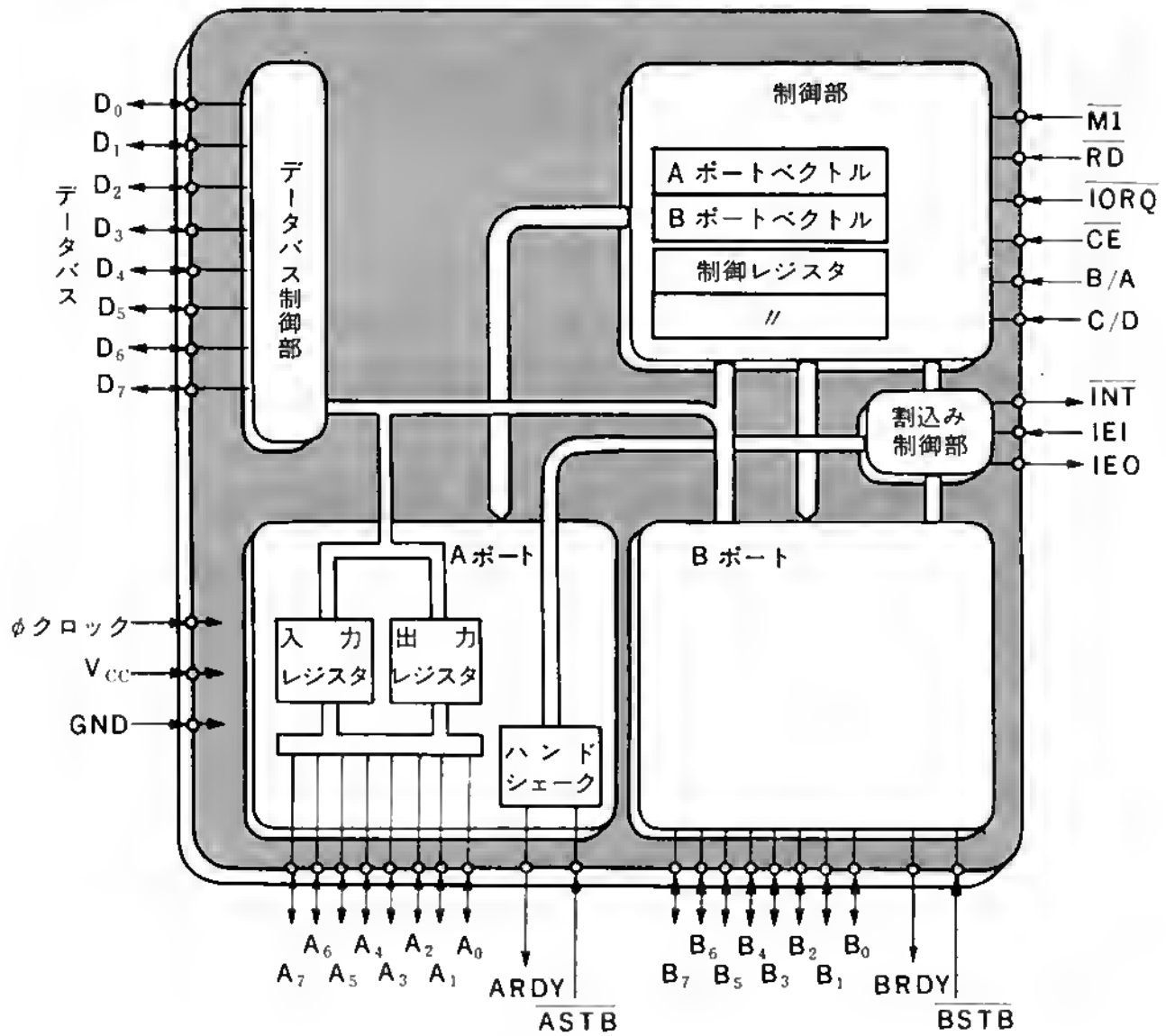
CPU の次によく使われるのが、この **PIO (Parallel Input/Output Interface Controller)** です。CPU からの信号を受けて、外部の装置に出力したり、外部装置からの信号を受けて、CPU へ伝えたりするデバイスです。

CPU の信号は、データバスを入出力以外の用途にも時分割で共用しているため、大変複雑な信号になっています。この中からある特定の時点のデータバス上の信号をとらえて外部との受け渡しをするためのもので、一般にラッチと呼ばれるロジックに、使いやすい機能をつけ加えた **LSI** といえます。信号は 8 ビット並列に扱います。動作は次の四つのモードがあります。

- |       |                            |   |                             |
|-------|----------------------------|---|-----------------------------|
| モード 0 | 出力モード                      | } | ハンドシェークにより 8 ビットデータを入・出力する。 |
| モード 1 | 入力モード                      |   |                             |
| モード 2 | 入出力モード                     |   |                             |
| モード 3 | ビットモード (ビット単位に入力、出力を選択できる) |   |                             |

モードやその他の動作条件は CPU からの信号により内部の制御レジスタに制御ワードを書くことによって設定されます。割り込みは CPU と PIO (または他のペリフェラル) のみで制御され、優先順位の決定や、どのペリフェラルからの割り込みかの解析や、割り込み処理プログラムルーチンからのリターンは、自動的に行なわれます。

PIO の内部にはほぼ同一のポートが二つあり、それぞれ別の動作モードで使うこともでき、割り込みも 2 系統発生させることができます。優先順位は、A → B の順です。



## 7 Z80 CTC

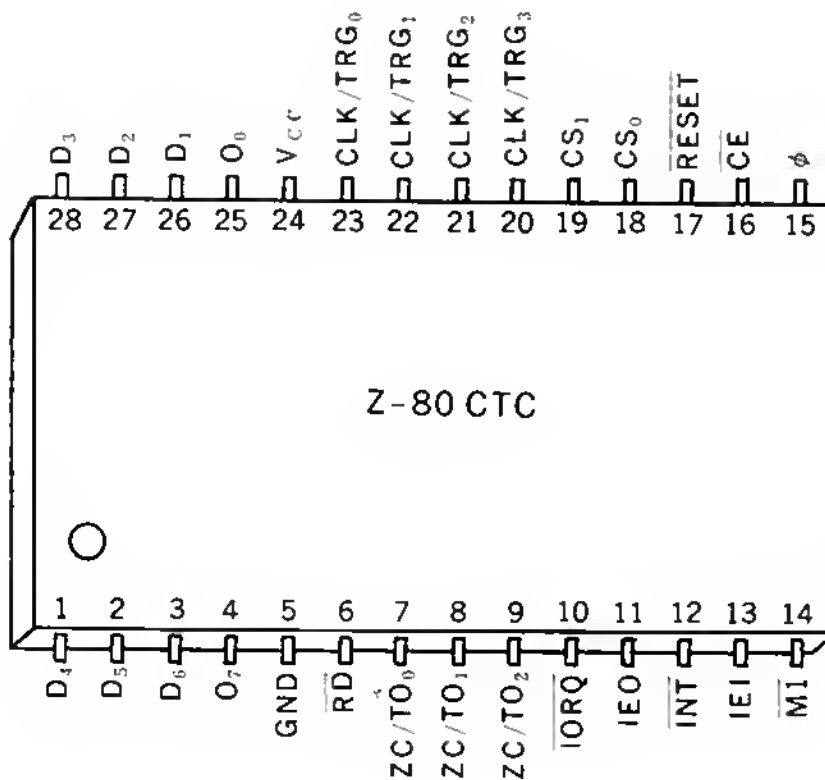
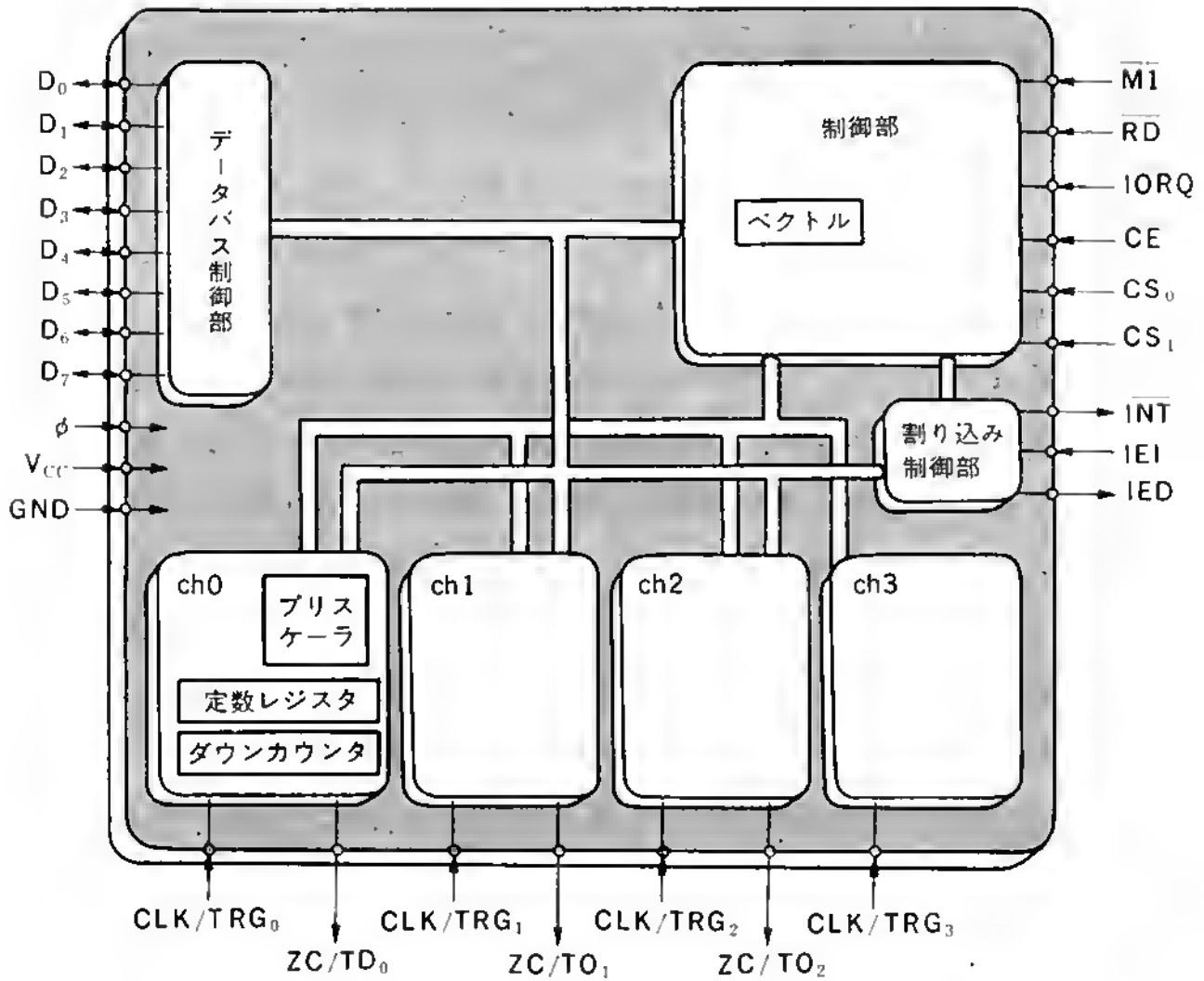
CTC (Counter Timer Circuit) は、パルスのカウントダウンをする LSI です。不特定周期の外部パルスをカウントし設定数になると割り込みを入れたり、ゼロカウント出力を出したりすることができます。不特定なパルスでなくクロックのような決まった周期のパルスをカウントすることにより、タイマとしても使えるわけです。

何個目のパルスで割り込みまたはゼロカウントするかの設定やカウントパルスのエッジ（立ち上りか立ち下りか）の選択などは CPU からの書き込みによってプログラムされます。またカウントの途中で残り数は CPU がカウンタの内容を読み出すことによって知ることができます。

設定できる数は 1 ~ 256 までですが、くり返えて何回目かの割り込みでカウント終了とすることにより範囲は広がりますし、ゼロカウント出力を次のカウンタへ入れてやれば、 $256^2$  までが扱えます。n 個つなげば  $256^n$  です。

1 個の CTC には四つのチャンネルが内蔵されていて、別々の目的に使用することができます。ただしピン数の関係で 4 個目のチャンネルは、ゼロカウント/タイムアウトの出力が出ていませんので、割り込みによりカウント終了となる使い方ができません。

システムクロックのカウントダウンによるタイマとしての使用時は、プリスケアラによってクロックを 16 または 256 分周したパルスをカウントします。タイマの起動は自動的に行なうことも、また外部トリガによって行なうこともできます。



## 8 Z-80 DMA

メモリ内、あるいはメモリと入出力ポートとのデータの転送は、通常は CPU が一度レジスタへ読み込み、次に書き出すという方式で行ないます。Z-80 には、単命令で一連の複数のデータを転送するブロック転送命令があります。プログラムステップ数は少なくなり、便利ではありますが、転送に要する時間は、1 バイトずつくり返しループを実行して転送するのと大差ありません。この転送の作業だけを高速に行なう周辺 LSI が **DMA (Direct Memory Access)** です。

CPU から、あらかじめ転送されるデータの入った元のアドレス（ソース）と転送先のアドレス（ディスティネーション）と、転送するバイト数を DMA 内のレジスタへ書き込めば、自動的に指定の転送を行ないます。ソースまたはディスティネーションは、自動的にカウントアップされますが、特定の IO ポートへ次々に出力または入力するときはカウントアップを止めておくこともできます。

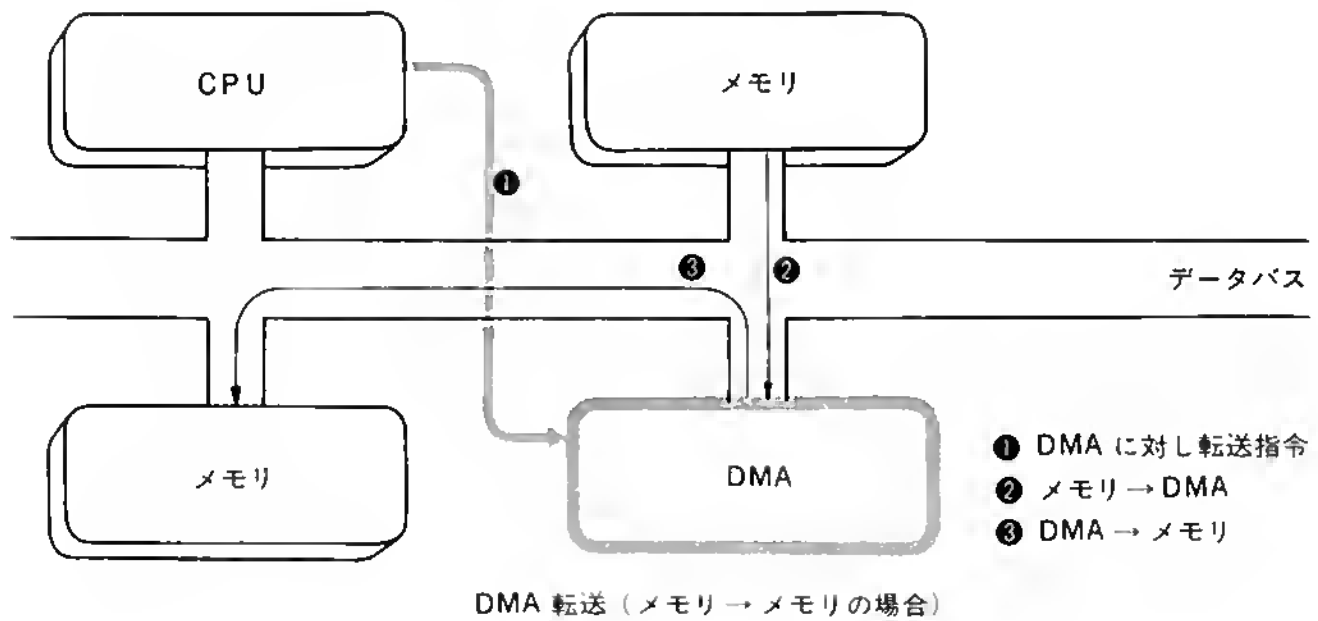
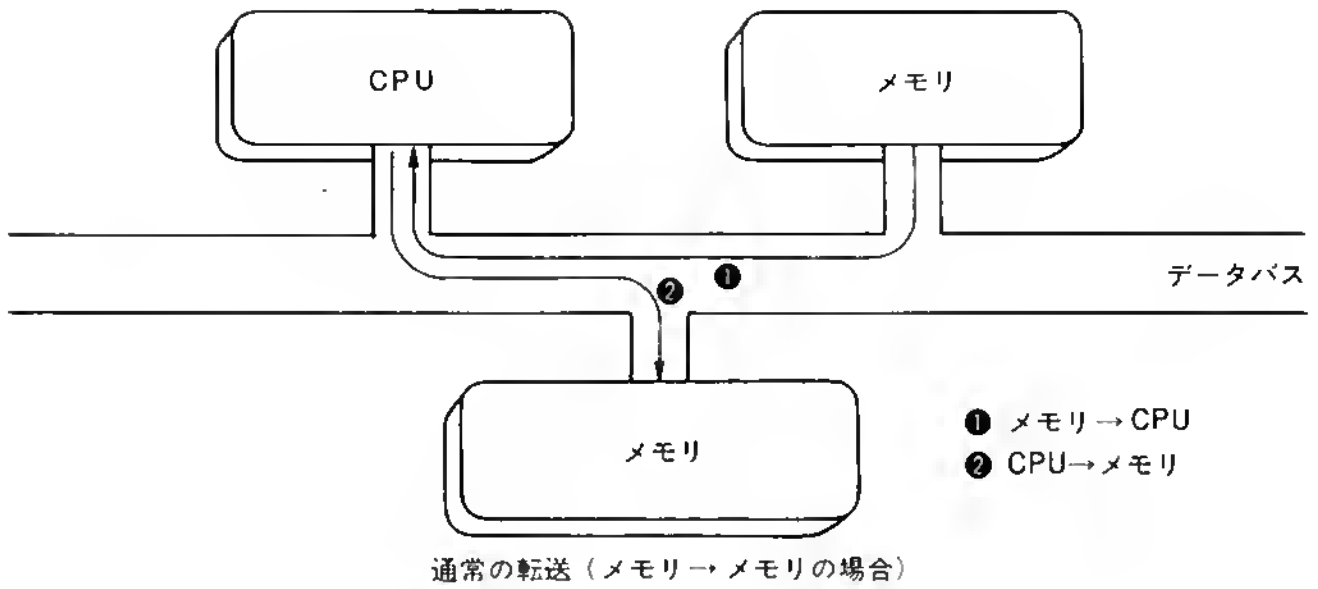
転送だけでなく、転送しながら、あるいは転送をせずに指定されたビットパターンとデータのビットパターンの一致をチェックすることもできます。これを**サーチ**と呼んでいます。サーチの終了（一致）は、転送終了とは区別できるよう割り込みがかかります。

DMA が動作中は、アドレスバスやデータバスは占有されますので CPU は待ち状態になりますが、指定により CPU 優先として一バイト単位に転送することや、外部ロジックからの切り替えにより CPU へバスを明け渡すこともできます。

DMA や SIO を使うシステムは、かなり大がかりなものになります。

本書では、初歩的な内容をわかりやすく解説することが主目的ですから、これらを対象からはずし、次のステップでの修得を期待します。





## 9 Z-80 SIO

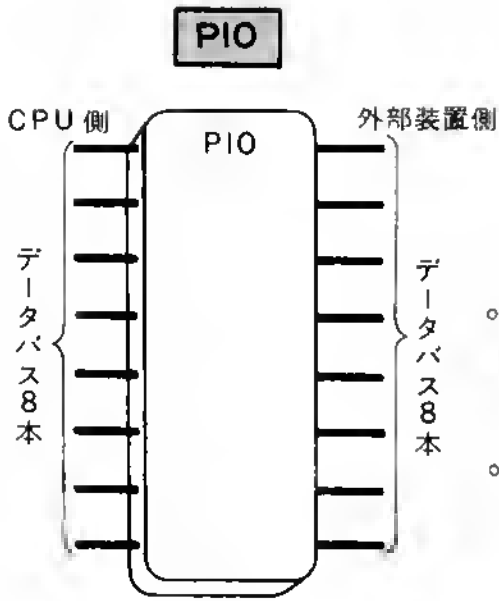
PIO は、データを並列に入出力するためのポートコントローラですが、この SIO (Serial Input/Output Controller) は、データを直列にすなわち時間経過に従って入出力するポートコントローラです。電話回線を使って長距離の通信を行なう場合はもとより、同一機器の筐体内でもケーブルが長くなる所は、データ通信線の数が少なくすむシリアル伝送を利用することがあります。もちろん他の条件が同一なら伝送速度はパラレルの  $1/8$  になることはやむを得ません。

直列データ伝送の方式には、同期・非同期、バイト指向・ビット指向などがあり、また通信速度や電圧、電流など、何種かの規格があります\*。コントロールプログラムと外部付加回路によってほとんどの方式に対応できる、きわめてぜいぜいな機能を持った LSI です。

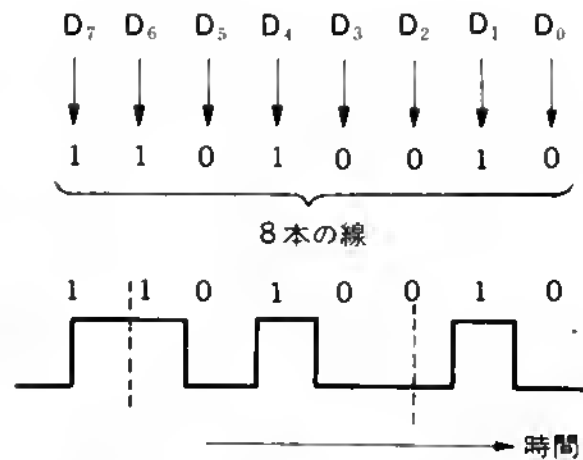
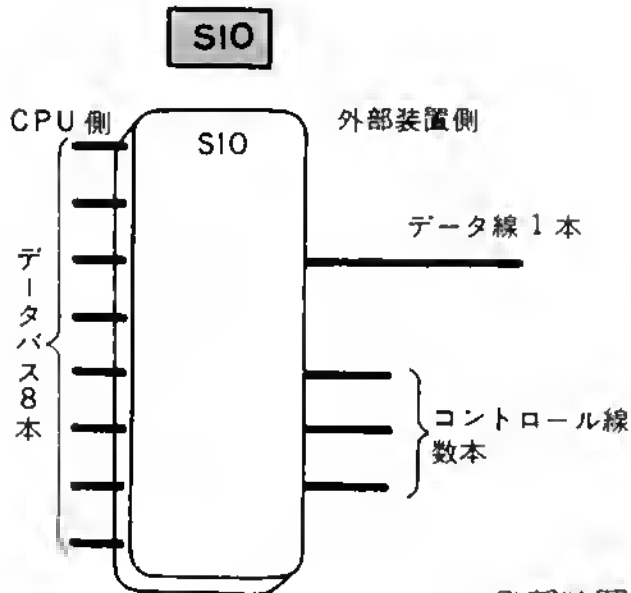
この SIO のチップ (LSI の中のシリコンウェハ上の本体) は 41 個の外部端子をもっていますが、パッケージは 40 ピンであるため、外部ピンに接続されない端子があるもの 2 種類と、2 本の端子を 1 本のピンに接続したものの合計 3 種類の SIO があります。ボンディング・オプションと呼んでおり SIO-0, SIO-1, SIO-2 と区別しています。

SIO はきわめて多機能であるため、すべての動作を理解しようとせず、シリアル伝送の方式を理解したうえで、SIO の機能の必要な部分を使用するよう考えるのが効果的です。

\* ジョン・E・マクナ马拉著：コンピュータ・データ通信技術、(CQ 出版)に詳しい。



- 外部装置から送り込まれるデータバスの内容のある瞬間をとらえて (IO サイクルに) CPU 側のシステムデータバスに送り出す。
- CPU 側のシステムデータバスの内容のある瞬間をとらえて保持し、外部装置への受け渡しのタイミングをとる。



- 外部装置から送られてくるデータ線上のパルス列を 8 ビットの並列データになおし CPU 側へ送り出す。
- 8 ビットの CPU 側データバスの内容を時間と共に順次データ線へ送り出す。
- 上記に必要なコントロール信号、内容の検査をするための付加情報を扱う。

## 10 メモリの種類と用途

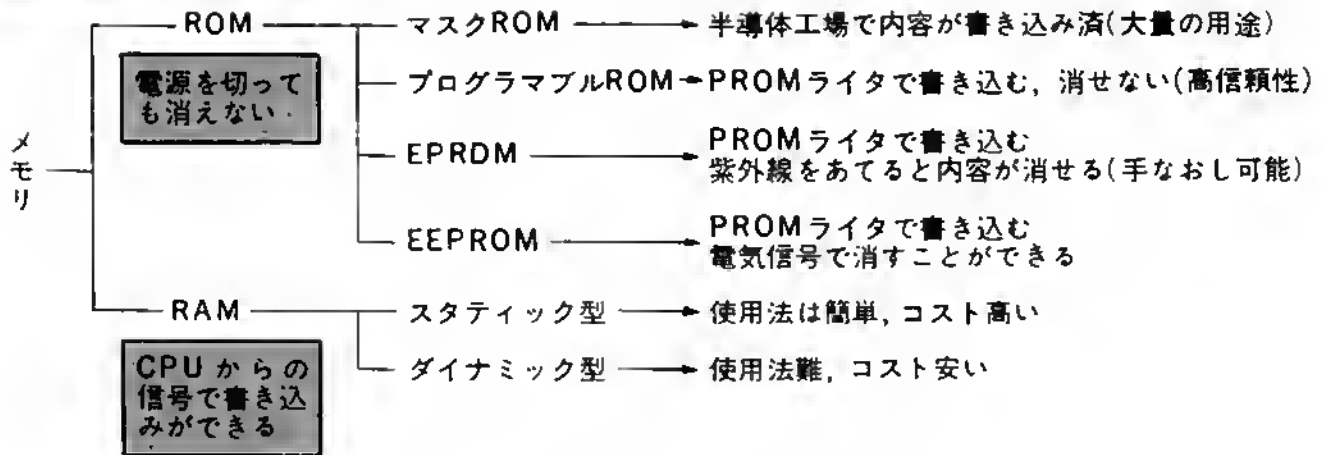
マイコンシステムで使われるメモリには何種類かありますが、現在ではほとんどが半導体素子で構成される LSI メモリです、磁性体メモリは読み書きが自由にできて、かつ電源を切っても内容が消えないのですが、価格やドライブ回路の点であまり使われません。

LSI メモリは ROM (リードオンリーメモリ) と RAM (ランダムアクセスメモリ) に大別されます、ROM は書き込みに特別な装置が必要で、CPU からのメモリライトはできません、ただし電源を切っても消えないため、プログラムを入れておくのに主に使われます、RAM は、CPU からの書き込みができますが、電源を切ると内容は消えてしまいます、

ROM には、マスク ROM と PROM (プログラマブル ROM) があります、マスク ROM は半導体工場で生産する時点で内容が決定されてしまいますので、同一内容の量産に向いています、PROM は、ROM ライタという装置で書き込みますが、一度書いたら変更できないもの (バイポーラ型 PROM) と、電気信号で消去できる EEPROM (エレクトリカルイレーザブル PROM) と紫外線照射により消去できる UVEPROM (ウルトラバイオレットイレーザブル PROM) があります、一般的には UVEPROM を単に EPROM と呼び多く使用されています、

RAM には SRAM (スタティック RAM) と DRAM (ダイナミック RAM) があります、SRAM はフリップフロップにより構成されたメモリで使いやすい特徴があります、DRAM はコンデンサの電荷蓄積を応用したメモリですから、時間と共に消えてしまいます、消えないうちに (実際には 2 ms 以下ごと) に一度読み出して再び同一内容を書き込むようにします、これを **リフレッシュ** といい、Z-80 では、CPU が命令コードを解析している時間を利用して自動的にリフレッシュする機能を持っています、コスト的にはメリットがあるのですが、ドライブのインターフェースが複雑になりむずかしさが残ります、最近ではメモリ内部にリフレッシュ機能を持ち、疑似的にスタティックと同様に使えるダイナミック RAM もあります、

## 使い方による分類



主に

ROMはプログラムを書いておく

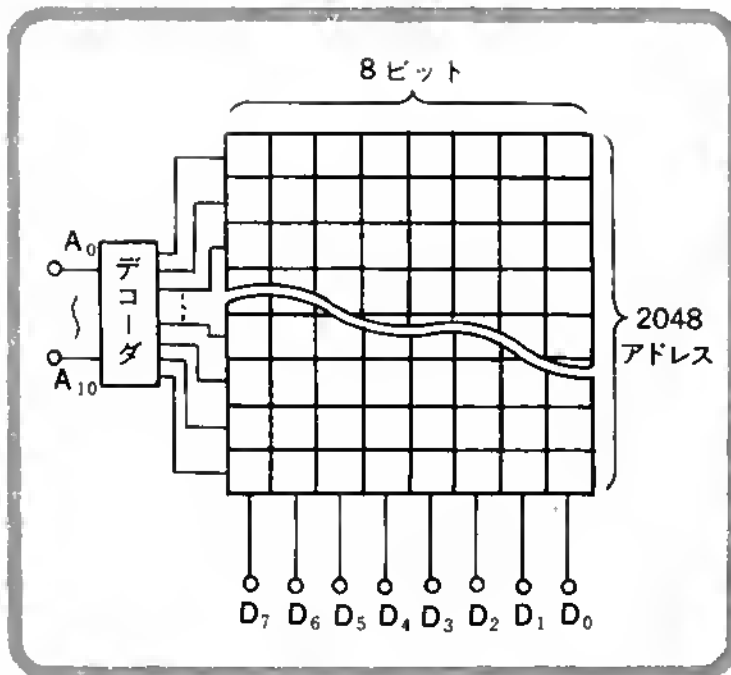
RAMは一時記憶用データ格納

読み出しだけだが電源を切っても消えないから

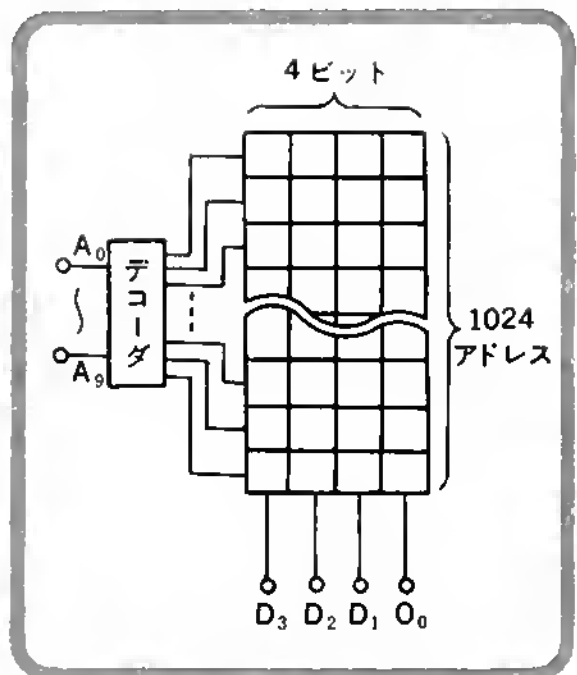
読み書きできるから

## よく使われるメモリの例

EPRDM (2716)



スタティック RAM (2114)



メモリは容量, 生産工程, パッケージなどが各種あります。  
メーカーのマニュアルによって適当なものをさがすことも重要な仕事です。

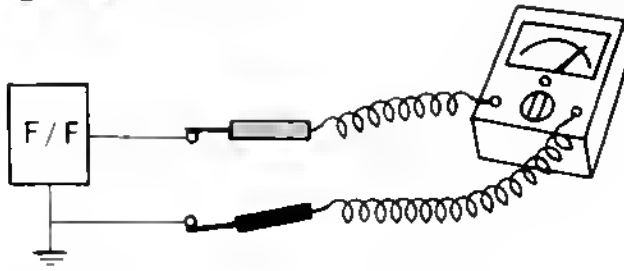
## ■ ■ ビットパターンと16進表現

CPU 内部や入出力信号線は、電氣的に電圧が高い状態と低い状態の二つで意味を持つようになっています。1本の電線では信号が有る、無い、の二つの状態でしかあり得ませんが、複数の電線を一束にして考えると、高低の組み合わせは2の累乗で増加します。アドレスバスは16本ありますが、16本の電線の高低の組み合わせは $2^{16} = 65536$ 通りになります。すなわち、CPUの出すアドレス情報は65536のメモリを識別できることになります。データバスは8本ですから、データとしては $2^8 = 256$ 通りの種類が得られます。しかし数回に分割してやりとりをすれば理論的には無限の組み合わせが得られます。8本では数値としては0～255までしか表現できないはずですが、実際にはもっと広い範囲の数値を扱えるのはこのためです。

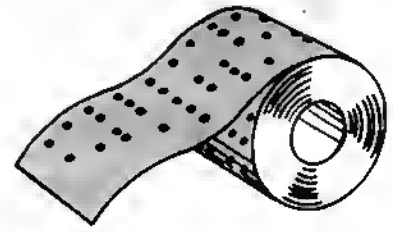
一つの2値状態を表現する単位を**ビット**と呼びます。アドレスバスは16ビットです。電圧の高い状態のことを**"H"**または**"1"**と呼び、低い状態のことを**"L"**または**"0"**と呼びます。16ビットの状態を表現するのに**"1"**と**"0"**を16個並べてもよいのですが、長くて扱いにくいので困ります。そこで**"1"**と**"0"**の組み合わせ(**ビットパターン**)を2進数とみなして、これを16進数に変換して扱うと大変便利になります。要するに、ビットパターンを4桁ずつ区切って**"1"****"0"**の組み合わせに名前をつけたと考えればわかりやすいでしょう。名前は数字の0～9、次が10でもよいのですが、2桁になってしまうのでAと呼び、次にB、C～Fまであります。0～Fで16ありますので16進数になるのです。本書でもビットパターンをいちいち書くのは大変なので、16進数で表現します。その場合は後にHをつけて10進数と区別します。Z-80のアセンブリ語でもこう書くきまりになっています。またマシン語の命令も本来はビットパターンで表現されるはずですが、一般的に16進数表現で扱います。

演算はビットパターンを2進数として行ないます。普通は0～255(8ビットでは)になりますが、符号を付けて-128～+127として考えることもできます。どちらをとるかでプログラムは異なります。

① 電圧が高い(低い)



② あながあいている(あいてない)



③ スイッチが上に倒れている(下に倒れている)



④ ランプが点灯している(消えている)



などを“1”というとき、( )内のときは“0”という → 誰かが決めた

※ 4つ要素が並ぶとすると下のとおり16通りのパターン(組み合わせ)が考えられる。

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

いちいち書くのは大変なのでパターンに名前をつけることにした(右欄)。

要素が4つだから →  $16通り = 2^4$

要素が8つだったら →  $2^8 = 256通り$

要素が16だったら →  $2^{16} = 65536通り$

※ パターンを2進数と見なすと名前は16進数になる。  
コンピュータ内の演算はこのパターンを2進数と見なすことにより数値を表現して行なわれるようになっている。

※ たった8本の電線(信号)でも256の状態を表現することができる。

- 1のことを“H”(high), 0のことを“L”(low)とよぶこともある。
- 普通は“0”になっていて、“1”になったら「信号がある」と決めておけば、“1”のことをアクティブという。
- 普通は“1”になっていて、“0”になったら「信号がある」と決めておけば、“0”のことをアクティブといい「この信号は負論理だ」という。

## 12 プログラムの実行

メモリに書き込まれているプログラムは、マシン語です。マシン語は英数字の組み合わせで、人間の目にはきわめてわかりにくいものですが、たとえば「AとBを加えてAに入れよ」とか、「止まれ」とか、「Aの内容を10番のポートへ出力せよ」といった命令にそれぞれ付けられた番号なのです。CPUは、日本語や英語で書かれるより、番号のほうが簡単に見分けることができます。

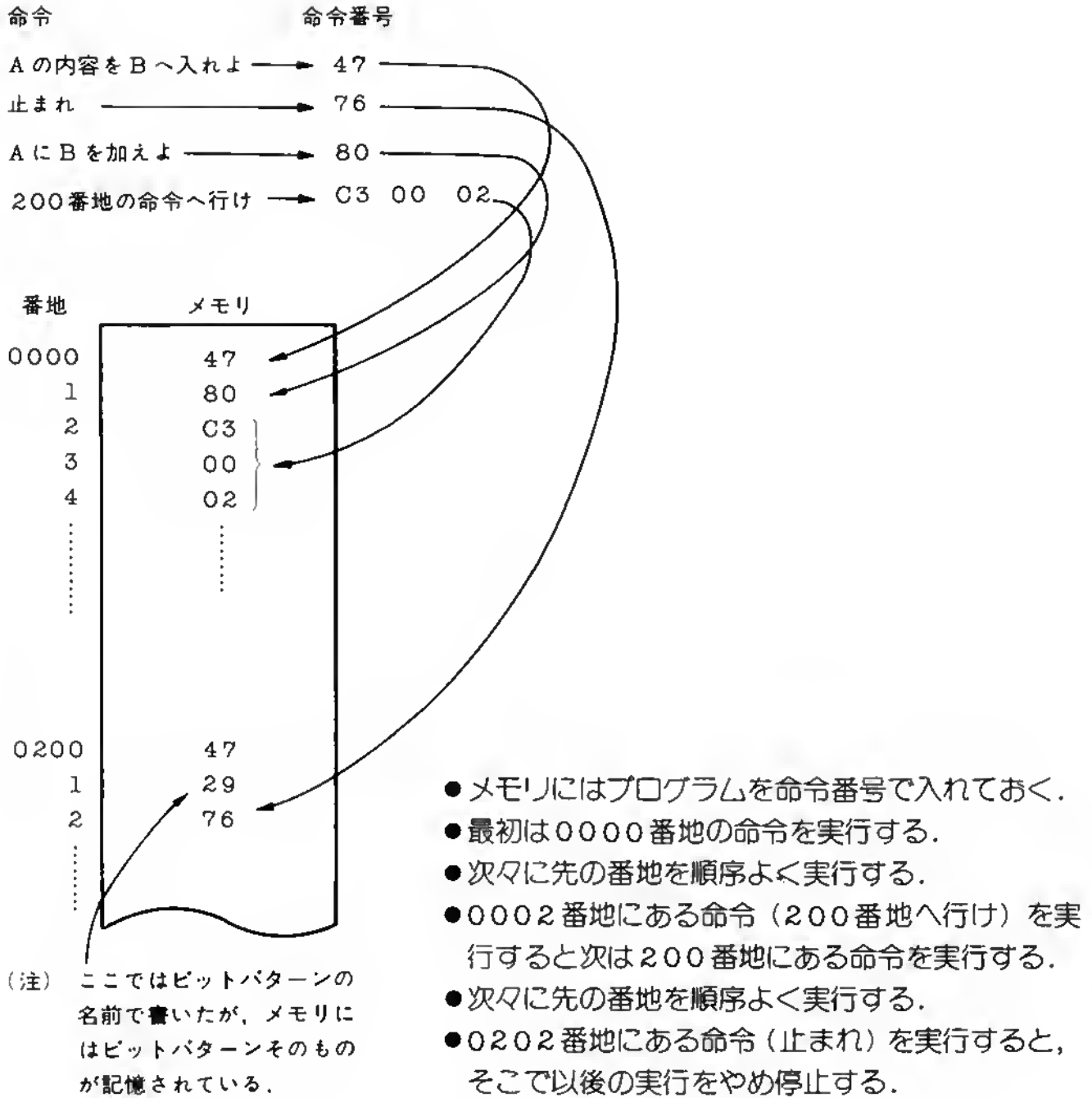
プログラムの書き込まれたメモリには、これも番号＝番地（アドレス）がつけられています。実行するときは、まずゼロ番地に書かれた命令から順次1ずつくり上げていきます。ところが命令の中に「何番地へ飛び」というのがあると、1ずつくり上げるのではなく、指定の番地へ飛び（ジャンプ）ます。また「前の演算の結果がゼロならば何番地へ飛び」という命令では、条件によって、ゼロなら飛び、ゼロでなければ次の番地の命令を実行します。

サブルーチン コール命令があると、指定された番地へ飛び、そこから順次実行し最後につけられたリターン命令で、先程飛んできた番地の次へ戻ることができます。同じ手順を何度も使いたいときによく使う方法で、飛ぶ前のプログラムを**メインルーチン**、飛んでくるプログラムを**サブルーチン**と呼びます。

サブルーチンとよく似ているのが割り込みです。割り込みは、命令があって飛ぶのではなく、電気信号がCPUの端子に与えられると、どこを実行中であってもある番地へ飛び、リターン命令で元のメインルーチンへ戻ります。**割り込み処理ルーチン**と呼ばれます。

CPU内には、プログラムカウンタ（PC）と呼ばれるレジスタがあり、この内容で、いま実行すべき命令の入っているメモリのアドレスを示すようになっています。一つの命令を実行するたびにこのPCを増やし、次の命令のアドレスを指します。また、特定のアドレスへジャンプするようなときは、PCへ飛び先のアドレスを強制的に入れることにより目的を達します。





## 13 CPUの信号のやりとり

CPUにはたくさんの信号入出力線が出ていますが、これらの信号線を使って基本的な六つの動作を時分割的に行ないます。それぞれを**マシンサイクル**と呼びます。

### 1. フェッチサイクル (M1 サイクル)

メモリに書き込まれているプログラム命令を CPU 内の命令解析用レジスタ (インストラクションレジスタ) へ読み込み、解読する。

### 2. メモリリードサイクル

メモリからデータを CPU 内のレジスタへ読み込む。

### 3. メモリライトサイクル

CPU 内のレジスタからメモリへ書き出す。

### 4. IO リードサイクル

入力ポートからデータを CPU 内のレジスタへ読み込む。

### 5. IO ライトサイクル

CPU 内のレジスタから出力ポートへデータを書き出す。

### 6. リフレッシュサイクル

ダイナミック RAM をリフレッシュするためのアドレス信号を出す。

以上のほかに一連のサイクル動作ではありませんが、単独の意味を持つ信号が七つあります。

### 1. ウェイト；クロックサイクル数を増やす。

### 2. インタラプト；割込要求。

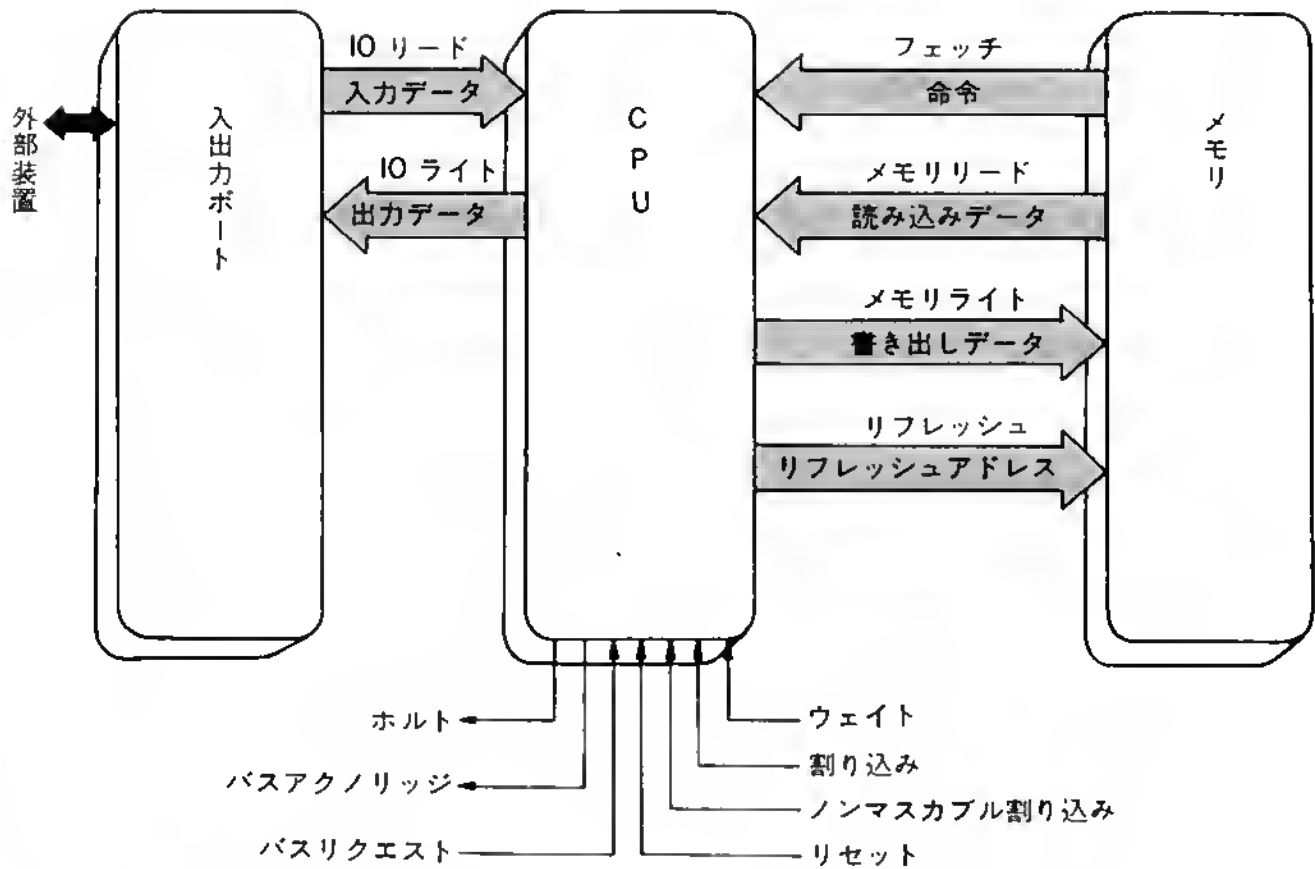
### 3. ノンマスカブルインタラプト；ノンマスカブル割込要求。

### 4. リセット；初期状態に戻す。

### 5. バスリクエスト；CPU の動作を停止させバスを空け渡す要求。

### 6. バスアクノリッジ；バスリクエストを受け付けた返事。

### 7. ホルト；CPU がホルト命令を実行し、停止状態に入ったことを外部へ知らせる。



➡ は一連の動作（マシンサイクル）で、アドレス情報とデータの受け渡しを伴う。

→ は情報の受け渡しは伴わない。

## ■ 4

## データバス、アドレスバスとシステム制御信号

CPUにある8本の**データバス**は、CPUと外部とのデータを出し入れするための信号線です。CPUの動作サイクルによって乗ってくるデータの意味は異なります。フェッチサイクルでは、メモリから命令コードが乗ってきますし、メモリアドレッシングサイクルではメモリからデータが乗ってきます。またメモリアドレッシングサイクルとIOアドレッシングサイクルでは、CPU内のレジスタからのデータがデータバス上に乗せられます。

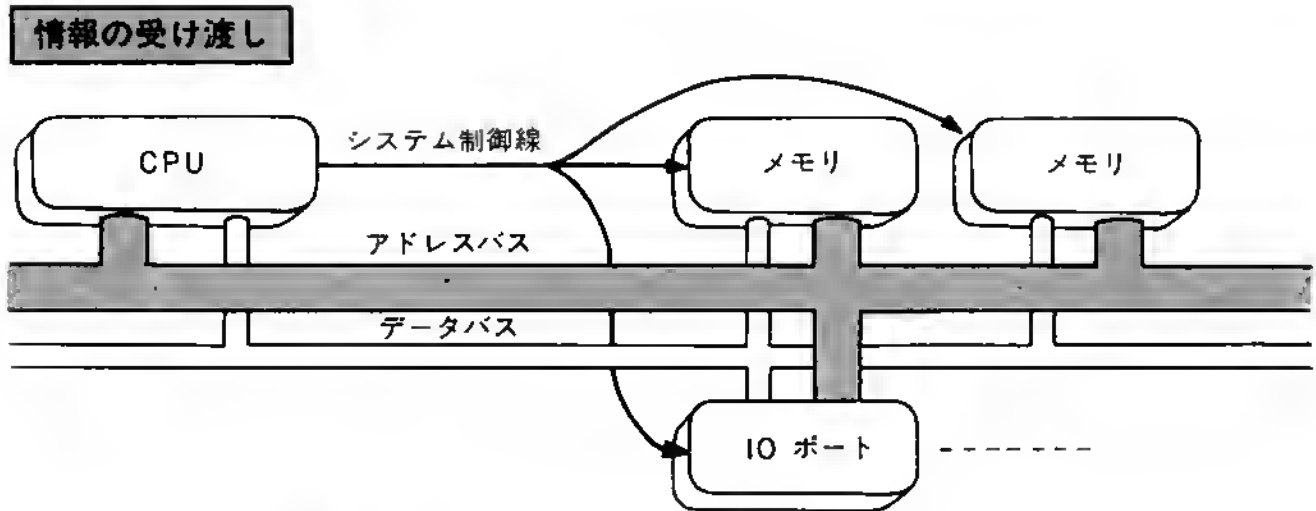
このようにデータバスは、時々によって種々の信号の出し入れに使われます。

また、**アドレスバス**もフェッチサイクルでは実行すべき命令の入っているアドレスが出力され、メモリアドレッシングサイクルでは、読んだり書いたりするメモリアドレスが出力され、また、IOアドレッシングサイクルでは、入出力すべきIOポートのアドレスが乗せられてきます。

データバスにせよアドレスバスにせよ目的の異なる信号が次々に乗せられますので、現在の信号は何を意味するかを識別するための別の信号線が必要になります。リード( $\overline{RD}$ )、ライト( $\overline{WR}$ )、メモリアクセス( $\overline{MREQ}$ )、IOアクセス( $\overline{IORQ}$ )、エムワン( $\overline{M1}$ )、リフレッシュ( $\overline{RFSH}$ )の各信号の組み合わせが、この役割をしています。実際にはこれらの各信号がすべて同時に出たり止まったりするのではなく、それぞれのタイミングで変化します。アドレスバス、データバスの占有時間も目的により異なります。

CPUの信号のうちデータバスとアドレスバスだけは正論理すなわち、1のとき“H”レベル、0のとき“L”レベルとなります。他の信号線はすべて負論理で、普段信号のないとき“H”になっていて、必要なときだけ“L”になります。

負論理の信号名の略称には——(バー)を付けて表わします。



システム制御信号線

エムワン ( $\overline{M1}$ )  
 メモリリクエスト ( $\overline{MREQ}$ )  
 IOリクエスト ( $\overline{IORQ}$ )  
 リード ( $\overline{RD}$ )  
 ライト ( $\overline{WR}$ )  
 リフレッシュ ( $\overline{RFSH}$ )

これらの組合せでアドレスバスに乗っている情報、データバスに乗っている情報あるいは、CPUがデータバスにどこから情報を乗せてもらいたいかを表現している。

※CPUは命令の内容によって、送り出すべき情報を送り出し、要求すべき情報を受けとる。

情報はデータバスを使う。

送り先や要求先は、システム制御信号とアドレスバスで指定する。

※メモリやIOポートなど周辺回路は、上のCPUの要求に対しある時間内に正確に応答しなければならない。これは汎用CPUの場合ユーザの利用技術にかかっている。

## 1.5 命令語の構成

マシン語の命令は、一つの機能を持つ命令が1～4バイトで構成されます。

1 バイト命令は、オペコードだけで意味を持つ命令です。

2 バイト命令は、オペコードと、オペランドに記述された数値を次の1バイトに持つものと、2バイトで一つのオペコードを意味するものがあります。

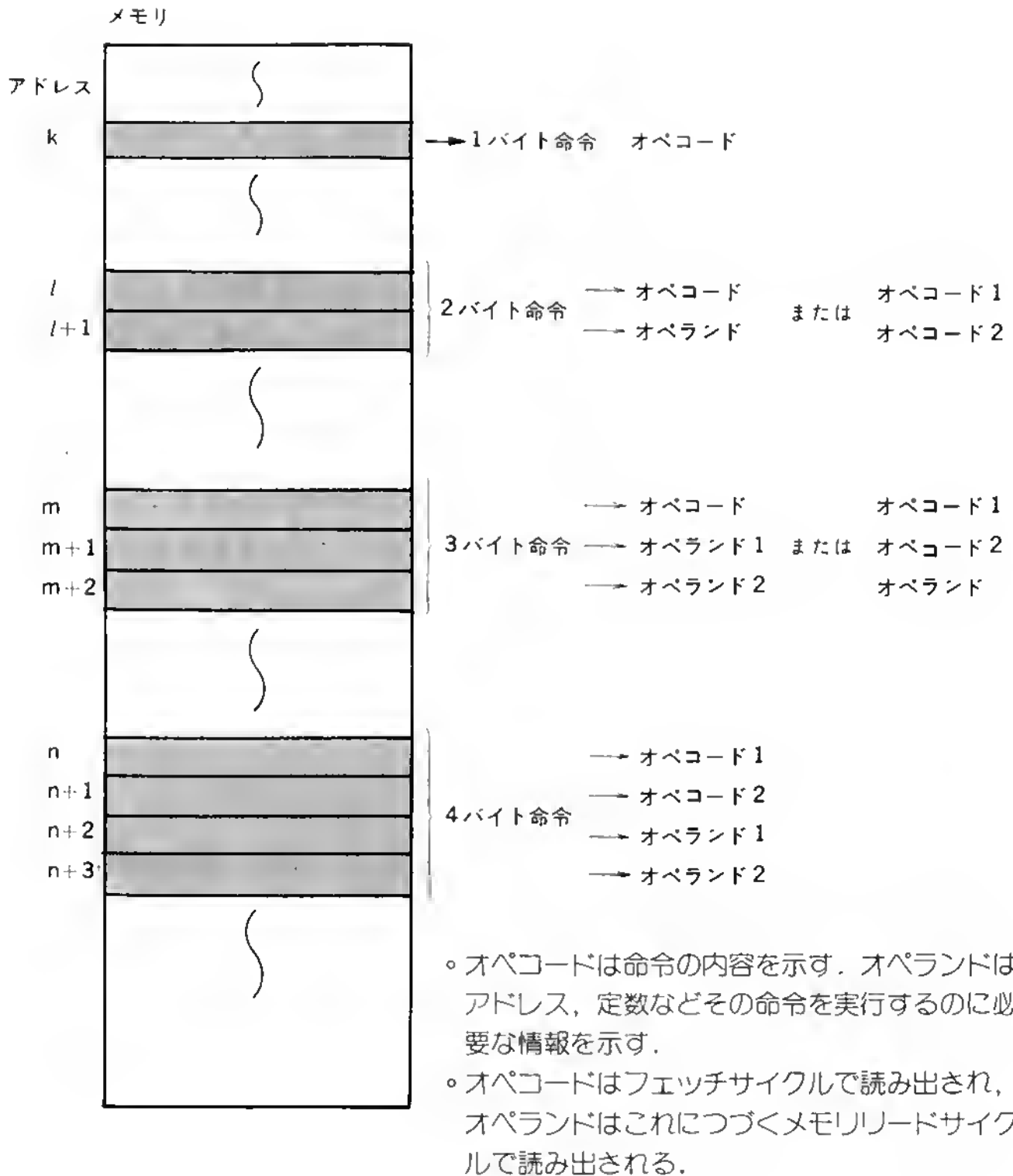
3 バイト命令は、オペコードと、オペランドに記述された数値を次の2バイトに持つものと、2バイトのオペコードと1バイトのオペランドを持つものがあります。

4 バイト命令は、2バイトのオペコードと2バイトのオペランド情報により構成されます。

オペコードが2バイトに渡る命令はZ-80特有のものです。最初の1バイトをフェッチし解読した時点で、2バイトのオペコードを持つことがわかりますから、あと1バイトをフェッチし解読します。したがってフェッチサイクルが二つあり、エムワン(M1)信号も2回出されます。

CPUはこの1～4バイトで構成される命令をフェッチサイクルとメモリーリードサイクルを起動して全部読み込み、意味を解析し実行します。これが終わると次の命令へと順次読み込み、解析、実行をくり返していきます。

一つの命令を実行する一巡を、**命令サイクル**（インストラクション サイクル）と呼びます。



## 16 命令の実行

例として [LD A, (mn)] という命令を分解してみましょう。

この命令には、3Aという番号がついていて、CPU 内部では、この番号で意味がわかるようにできています。3A の次に二つの情報、n と m が連続している 3 バイト構成の命令です。具体的には「mn 番地のメモリの内容を CPU 内の A レジスタへ入れよ」という意味があります。m と n は各 8 ビットで、メモリアドレスは 16 ビットで表わされますので、二つで一つのアドレス情報になります。このときメモリ上には m (上位) と n (下位) を反対にして n, m の順で並べる規定になっています。

### M1 サイクル (フェッチサイクル)

プログラムカウンタ (PC) の内容で示される番地から 3A をフェッチする CPU 内のインストラクションレジスタへ入れてこの意味を解析すると、次に何をすべきかわかり以下を実行する。PC に 1 を加える。

### M2 サイクル (メモリリードサイクル)

PC の内容番地より n を読み込んで制御部のレジスタへ一時格納する。  
PC に 1 を加える。

### M3 サイクル (メモリリードサイクル)

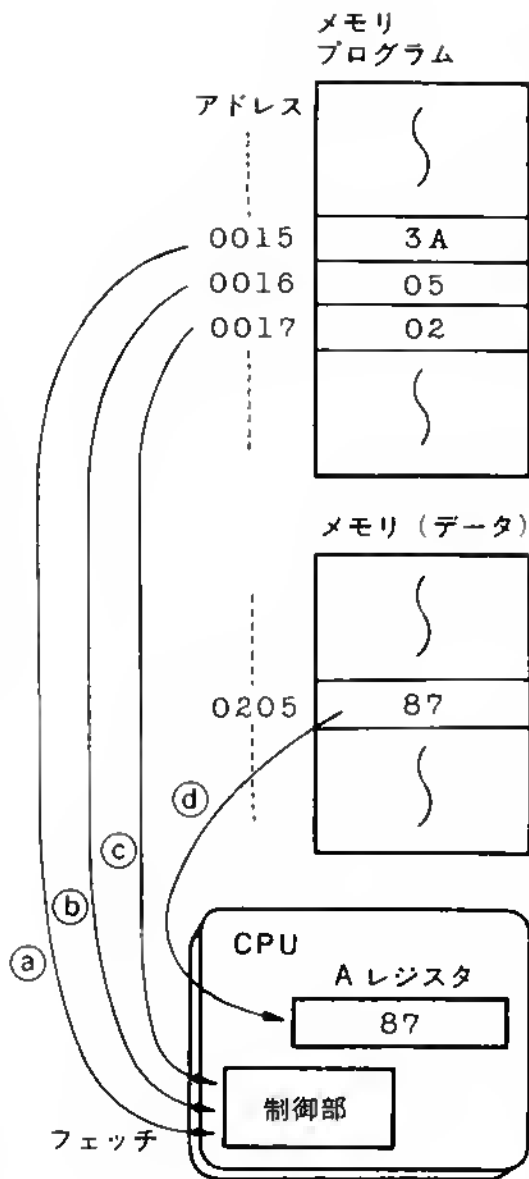
PC の内容番地より m を読み込んで制御部のレジスタへ一時格納する。  
PC に 1 を加える。

### M4 サイクル (メモリリードサイクル)

制御部のレジスタへ格納した n と m により一つのアドレス mn を構成し、mn 番地の内容を読んで A レジスタへ入れる。

以上の四つのマシンサイクルによって命令を実行したわけですが、命令の内容によって、それぞれのサイクルの組み合わせは異なり、サイクル数も異なります。ただし、どんな命令でも必ず、フェッチサイクルが最初に起動され、フェッチした命令の内容によって次に何をするか決定します。





## 命令実行の一例

- ① 0015 番地の命令を実行する順番がくると、CPUはアドレスバスに0015を乗せ、システム制御信号はメモリからの命令読み込みを要求する。
- ② メモリはこれに対し0015番地の中身をデータバスへ乗せる。
- ③ CPUはこれを受けとり命令の意味を解析する。

— 以下解析の結果 —

- ④ CPUはアドレスバスに0016を乗せ、システム制御信号によりメモリからの読み込みを要求する。
- ⑤ メモリはこれに対し0016番地の中身をデータバスに乗せる。
- ⑥ CPUはこれを受けとる。
- ⑦ 同様に0017番地の中身も受けとる。
- ⑧ いま読み込んだ0016と0017番地の中身をつないで0205という値を作り、これをアドレスバスに乗せ、システム制御信号によりメモリからの読み込みを要求する。
- ⑨ メモリはこれに対し0205番地の中身をデータバスに乗せる。
- ⑩ CPUはこれを受けとりAレジスタへ入れる。

(①～③がフェッチサイクルで各命令実行ごとにまずこのサイクルに入る。)

## 17 アセンブラ記法のルール 17

16進数で表現されるマシン語命令では、人間には覚えにくくわかりづらいため、直感に訴えるような英語の略語（ニモニック）を対応づけたのが**アセンブリ言語**です。アセンブリ言語で書かれたプログラムを**ソースプログラム**と呼びます。ソース プログラムをマシン語に自動的に置き換えるプログラムが**アセンブラ**です。このアセンブラ プログラムが読み込むソース プログラムの書き方には、一定のルールがあります。

ラベルはアドレスに付ける仮の名称です。この命令のアドレスへジャンプしたり参照したりするために必要なところだけでよく、つけなくてもよいのです。英文字で始まる6文字以内の英数字列で後で見てわかりやすい名前を付けます。

オペコードは命令語のニモニックです。CPUの持つ命令の中から必要なものを選んで使います。

オペランドは、オペコードによってはないものもあります。一つだけのものも、`“ , ”`で区切って二つ必要なものもあります。二つ必要な命令の場合は、前に書くのがディスティネーション、後に書くのをソースといって、ソースからディスティネーションへのデータの移動を意味します。ソースまたはディスティネーションをカッコでくくるときは、カッコ内のアドレスのメモリに入っている内容を意味します。カッコ内がレジスタ名の場合は、そのレジスタに入っている内容をアドレスとしたメモリを意味します。

コメントはプログラムをあとで見たときにわかりやすくするために必要なことをメモしておく所です。的確なコメントの豊富につけられたプログラムは、誰が見てもよくわかり、大変便利なものです。

アセンブリ言語で書く命令には、上のようにマシン語命令に変換されるもののほかに、疑似命令といって、マシン語に変換されずに、アセンブラ プログラムに対して指示を与えるだけの命令もあります。また、アセンブラ プログラムがマクロ アセンブラと呼ばれる場合は、一命令をあらかじめ別に定義された数個以上のマシン語命令群に変換するマクロ命令といったものも使うことができます。

		ラベル			
		ニモニック			
		ディスティネーション			
		ソース			
		オペランド			
LOOP	LD	A, B	この定数が16進表現であることを明示する。		
	ADD	A, B	Bレジスタの内容をAレジスタへ入れよ		
	LD	HL, 0802H	AレジスタへBレジスタの内容を加えよ		
	LD	HL, (0802H)	HLレジスタへ定数0802Hを入れよ		
	OUT	(00H), A	HLレジスタへ0802番地のメモリの内容を入れよ		
	IN	A, (00H)	00H番地のIOポートへAレジスタの内容を出力せよ		
	JP	LOOP	Aレジスタへ00H番地のIOポートから読み込め		
JOB1	LD	(HL), A	LOOP番地へ飛び、その命令以下を実行せよ		
			HLレジスタの内容をアドレスとするメモリへ		
			Aレジスタの内容を書き出せ		
	END		プログラムの終了をアセンブラプログラムへ知らせる		

オペランドの表記法

	ディスティネーション	ソ ー ス
レジスタ名	レ ジ ス タ	レ ジ ス タ
定 数	—	定 数
(レジスタ名)	レジスタの内容アドレスのメモリ	レジスタの内容アドレスのメモリ
(定 数)	定数アドレスのメモリ	定数アドレスのメモリ

16進表現で、最上位がA～Fのアルファベットになるときは頭に0を付け、ラベル名と区別する。

(例) A000Hは0A000Hと書く

## 18 アセンブラ記法のルール - 2

行の先頭、すなわち前の行の最後につけられた C/R (キャリッジリターン) の次に続く文字列は、**ラベル**です。英大文字に続く英数字の組み合わせで、6 文字以上は無視されます。ラベルの文字列の直後に “:” コロンがあれば、行頭でなくてもラベルとみなされます。ラベルをつけない行は、1 個以上のスペースをラベルの代わりに入れておきます。

ラベルと 1 個以上のスペースで区切られた後にオペコードを書きます。命令表の中から選んで下さい。全部で 74 あります。

オペコードと 1 個以上のスペースで区切られた後には、そのオペコードに対するオペランドが続きます。二つ必要なときは “,” コンマで区切ります。このオペランドは、レジスタ名、16 進表現の定数、10 進表現の定数、フラグの状態 (分岐の条件) 名、を書きます。定数は、他の命令に付けたラベルを書いてもよく、ジャンプする場合などは特に絶対番地を計算しなくてもよいので便利です。オペランドのソースやディスティネーションに ( ) カッコを付けたときは、その内容を番地とするメモリの内容を意味します。

オペランドに書く定数は、10 進表現で書いてもよいし、16 進表現で書いてもよいのですが、16 進表現で書くときは、後に H (ヘキサデシマルの頭文字) をつけておきます。また、先頭がアルファベット A ~ F で始まるときはさらに前に 0 をつけてラベルでないことを特徴づける約束になっています。

コメントは行中のどこにでも “;” セミコロンがあれば、それ以後は自由にコメントエリアとして使うことができます。ただし 1 行は通常 80 字位までで切られることがありますので、長くなる場合は数行に分けます。

Z-80 のアセンブリ言語は標準的な取り決めがありますが、アセンブラ プログラムの種類 (メーカーや機種) により若干の違いがあります。アセンブラを使うときは、それぞれの説明書を一読して下さい。

コメント

Z80 MACRO ASSEMBLER V1.1 PAGE 1

```

1      ; IPL
2      ORG *      0FF00H
3  FF00 3EE0      LO      A, 0E0H      ; MOOE
4  FF02 D3F3      OUT     (0F3H), A      ; MOR0
5  FF04 3E37      LD      A, 037H      ; SLP
6  FF06 D3F2      OUT     (0F2H), A      ; COR
7  FF08 DBF6      LP0:    IN      A, (0F6H)      ; ISR
8  FF0A CB7F      BIT     7, A      ; CCE
9  FF0C 20FA      JR      NZ, LP0      ; CCE WAIT
10 FF0E 0610      LD      B, 16      ; BLOCK COUNT
11 FF10 210000     LD      HL, 0000H      ; ADDRESS
12 FF13 3EFF      LP1:    LD      A, 0FFH      ; WDC SET
13 FF15 D3F1      OUT     (0F1H), A      ; WDC
14 FF17 3E38      LD      A, 038H      ; ROL
15 FF19 D3F2      OUT     (0F2H), A      ; COR
16 FF18 DBF6      LP2:    IN      A, (0F6H)      ; ISR
17 FF1D CB6F      BIT     5, A      ; DA
18 FF1F 2007      JR      NZ, LP3      ; 定数(番地)の代わりにラベル
19 FF21 DBF0      IN      A, (0F0H)      ; DBR で指定
20 FF23 2F        CPL
21 FF24 77        LD      (HL), A      ; DATA LOAD
22 FF25 23        INC     HL      ; ADDRESS COUNT UP
23 FF26 1BF3      JR      LP2
24 FF28 CB7F      LP3:    BIT     7, A      ; CCE
25 FF2A 20EF      JR      NZ, LP2      ; CCE/DA WAIT
26 FF2C 10E5      DJNZ    LP1      ; ENO?
27 FF2E C300E0     JP      EXIT      ; RETURN TO MONITOR
28      ; コメント行 (見やすくするために一行あける)
29      E000      EXIT:    EQU *      0E000H
30                      END *

```

ラベル

オペコード

オペランド

マシン語命令  
(オブジェクト)

ソースプログラム

↑ タイプライタキーを打って入力する

アドレス

↑ アセンブラが自動的に変換する

行番号

- \* ORG : 以下のプログラムを配置するアドレスを指示する。  
 EQU : ラベル名の文字列をオペランドの数値とすることを指示する。  
 END : プログラムの終りを明示する。

## 19 命令の種類

機能別に命令を分類すると、次のようになります。

### 8ビットロード

1バイトの情報をレジスタ間、レジスタとメモリ間で転送する。1バイト定数をレジスタかメモリへ入れる。

### 16ビットロード

2バイトの情報をレジスタ間、レジスタとメモリ間で転送する。2バイト定数をレジスタへ入れる。

### レジスタ交換

レジスタの内容を入れ替える。

### メモリブロック転送

メモリ内の複数バイトのブロックを別のアドレスへ転送する。

### メモリブロックサーチ

メモリ内ブロックに指定の情報があるかどうか探す。

### 8ビット演算、論理演算

1バイト単位の加減算、論理和、論理積、排他的論理和、カウントアップ、カウントダウン、比較をする。1または2の補数をとる。

### 16ビット演算

2バイト単位の加減算、カウントアップ、カウントダウンをする。

### ローテートシフト

レジスタ、メモリのビットパターンを回転させまたは左右へずらす。

### 10進補正

2進法10進数として扱うデータの加減算後の補正をする。

### フラグ操作

キャリフラグを"1"にする。反転させる。

### CPU制御

プログラムの実行と、割り込みを制御する。

**ビット操作**

レジスタ、メモリの特定の1ビットを"0"か"1"にする、また、判定する。

**ジャンプ**

プログラムの実行を条件によりまたは無条件で、指定のアドレスへ移行させる。

**コール、リターン、リスタート**

プログラムの実行を条件によりまたは無条件で、指定のアドレスへ移行させる、ただしこのとき、リターン命令によりもとのアドレスへ戻れるような手順を含んでいる。

**入力出力**

指定のIOポートへレジスタとの間で1バイト情報を入・出力する。

**連続入出力**

メモリブロックを1バイト単位に連続して出力する。

メモリブロックへ1バイト単位に連続して入力する。

**マイナス数の表現(2の補数)**

2進数でマイナスを表現するには、多くの場合次の手法がとられます。

4ビットだけで考えれば、ゼロすなわち0000のひとつ前の-1は1111になります、なんとなればこれに1をたすと桁上りをして10000ですが、4ビットだけで考えているので桁上りを無視せざるを得ないからです。

0	0000	8	1000	→ -8
1	0001	9	1001	→ -7
2	0010	A(10)	1010	→ -6
3	0011	B(11)	1011	→ -5
4	0100	C(12)	1100	→ -4
5	0101	D(13)	1101	→ -3
6	0110	E(14)	1110	→ -2
7	0111	F(15)	1111	→ -1

上記のとおり、4ビットでは0~F(10)または-8~+7を表現することができます。正数を負数に変えるには、ビットの"1""0"を反転させて1を加えればよいのです。

## 20 メモリ空間とIO空間

Z-80 では（他の 8 ビット CPU もほとんど）接続できるメモリは最大 64 K バイト（K は 1024）です。これはアドレスバスが 16 本あるので  $2^{16} = 64\text{ K}$  となるからにほかなりません。メモリとは別に IO の領域として 256 のアドレスがあります。したがって IO ポートは 256 まで接続できることになります。これは、インプット、アウトプットのサイクルではアドレスバスの下位 8 ビットのみ使用しているためで、 $2^8 = 256$  ということです。同じアドレスバスをメモリリクエスト信号と IO リクエスト信号により切り替えて使用するためにメモリ空間と IO 空間を別々に持ったことになります。

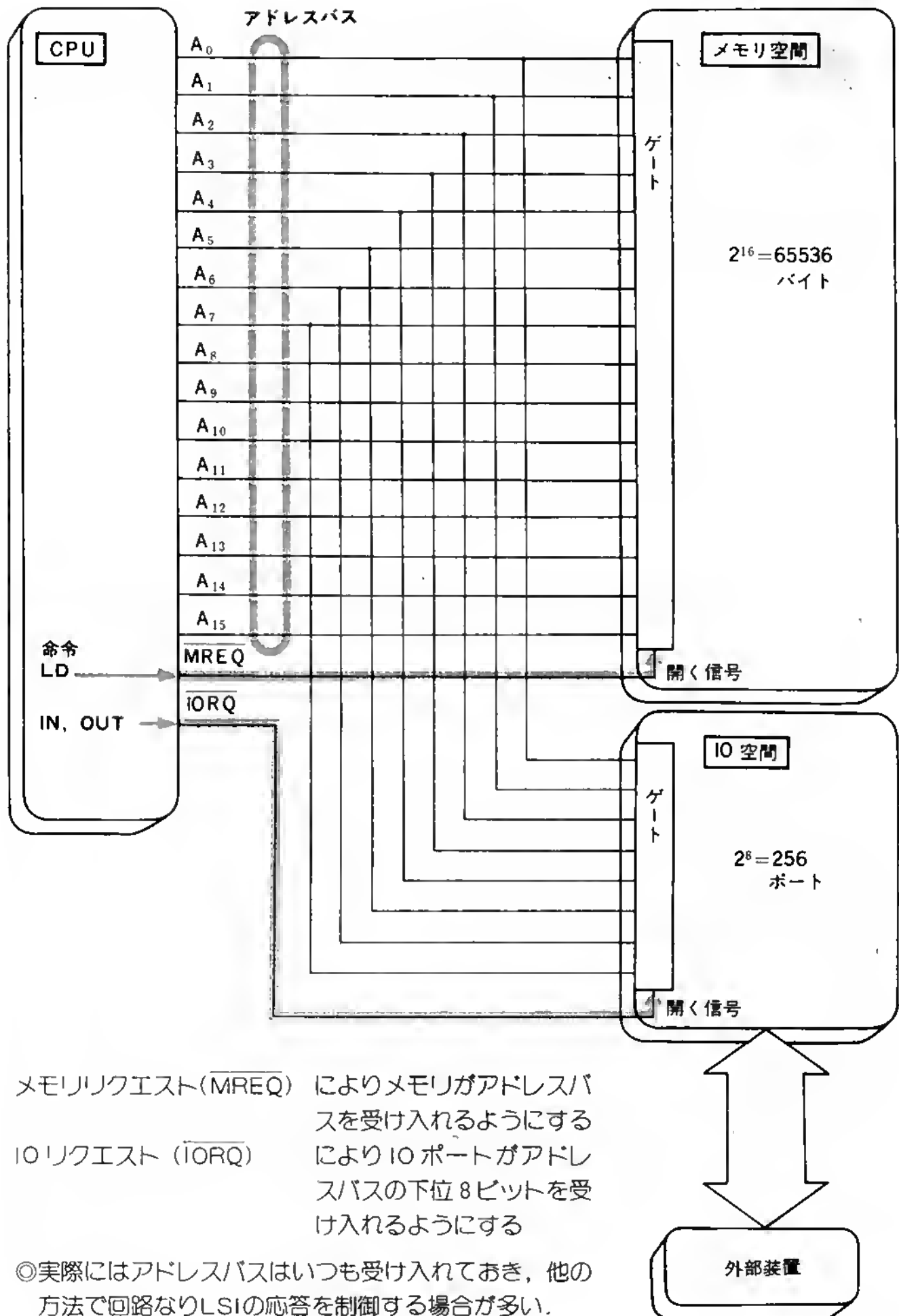
メモリ空間へのアクセスは、ロード命令を使い、IO 空間へのアクセスはイン、アウト命令を使います。IO 空間へのアクセスのほうが、タイミング的に長い時間がとれるようになっています。

I7 項の説明では省略しましたが、IO 空間のアドレスを示す場合も OUT（OOH）、A のようにカッコに入れるきまりになっています。

**バイト** ○8 ビットを 1 バイトと呼ぶ。8 ビットの CPU はバイトマシンと呼ばれ、8 ビットを同時に扱うので、メモリは 1 アドレスのアクセスで 8 ビット（データバス 8 本）の読み書きが行なえる構成とする。

○したがって、1 アドレス 4 ビットのメモリ（2114 など）では 2 個、1 アドレス 1 ビットのメモリ（4116 など）では 8 個並列に並べる。





## 21 アドレスデコーダ

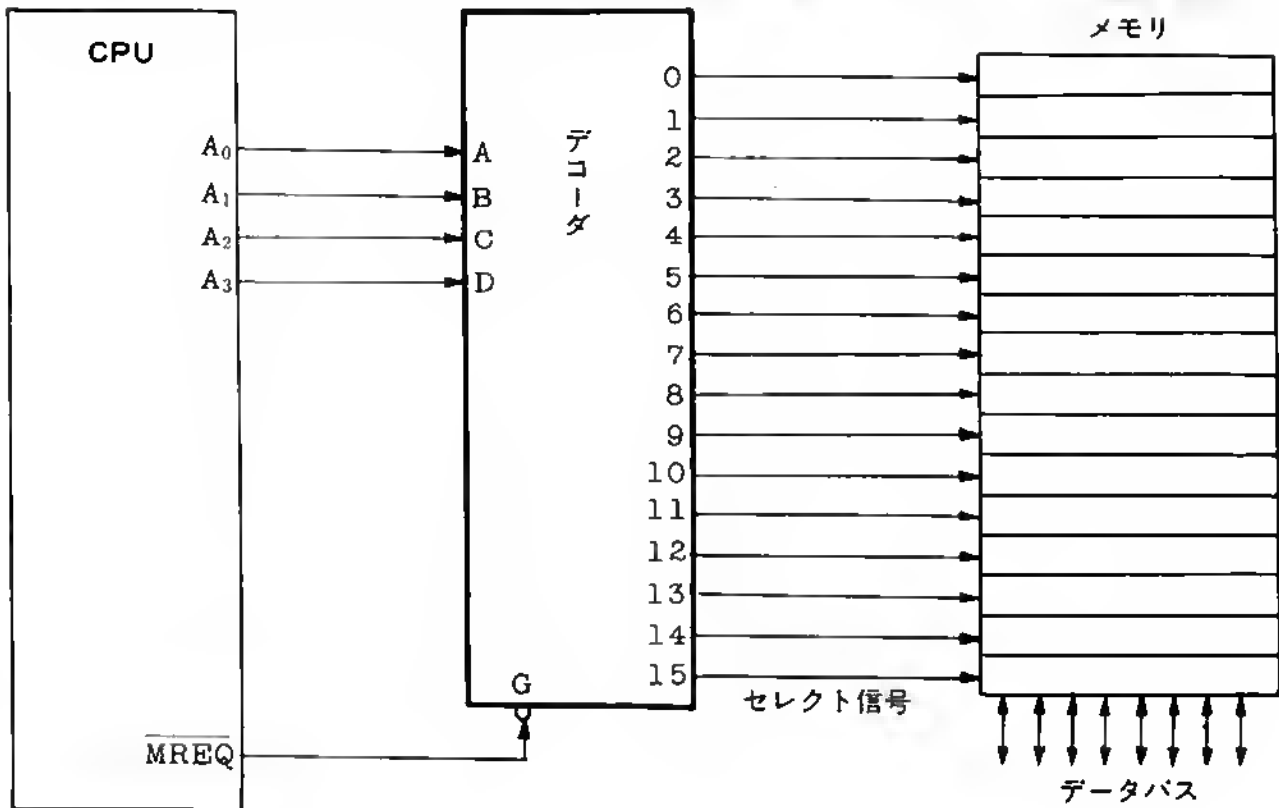
アドレスバスに出力されるビットパターンは 16 本の端子から 65536 通りもありますから、どのパターンのときはどのメモリを選ぶかを決めてやらなければなりません。

アドレスバスに出てくるアドレス信号のビットパターンから特定のメモリや IO ポートを選び出すセレクト信号(イネーブル信号)を作り出すわけです。メモリはたとえば 256 バイトのメモリであれば、LSI の中に 256 バイトすなわちアドレスバス 8 本分のデコーダを持っていて、内部で特定のメモリセルを選択してくれます。ところが、このメモリを複数使用するときや、IO ポート用のペリフェラルを複数使用するときは、CPU とペリフェラルやメモリの間にアドレスデコーダが必要になります。デコーダは通常 TTL 等の論理 IC で構成します。

番地だけでなく、メモリ空間と IO 空間の切り替えも、ここで行なうのが普通です。メモリリクエスト ( $\overline{\text{MREQ}}$ ) がアクティブつまり負論理出力ですから "L" になったときはメモリが、IO リクエスト ( $\overline{\text{IORQ}}$ ) がアクティブならば IO ポートがイネーブルになるようなロジックを組むのです。通常問題になることは少ないのですが、メモリや IO ポートへセレクト信号を出してから実際にイネーブルになり、データを受け入れまたは出力するまでの時間は、CPU の動作時間に適合しなければなりません。CPU より遅くて間に合わない場合はウェイト信号を CPU に与えて一時待たせるなどの方法をとります。メモリや IO の応答時間と、アドレスデコーダなどの遅れ時間を含めて考える必要があります。しかし一般には、この遅れは無視できる場合のほうが多いようです。

アドレスデコーダとしてよく使われる TTL は 74I39 や 74I55 などがあります。TTL の規格表から目的に合うものを探し出し、価格や入手状況を検討して決定して下さい。

(例) 16 バイトメモリのアドレスデコーダ

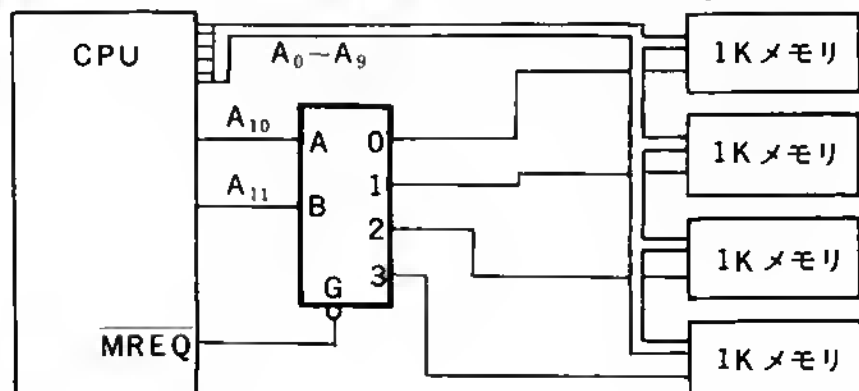
16 バイト =  $2^4$  : 4 本のアдресバスで 16 のアドレスを表現できる

## デコーダの出力

(X は何でもよい)

G	D	C	B	A	出力
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	X	X	X	X	ナシ

- MREQ がノンアクティブ (負論理だから "1") のときはメモリセレクトはしない。
- MREQ がアクティブ ("0") のときだけ  $A_0$  から  $A_3$  のビットパターン16種により16本の出力のどれかへセレクト信号を出す。
- セレクトされたメモリはデータバスとデータの受け渡しを行なう。

(参考) 1Kバイトのメモリ4個を使ったときのデコーダ  $\left[ \begin{matrix} 1K = 1024 \\ = 2^{10} \end{matrix} \right]$ 

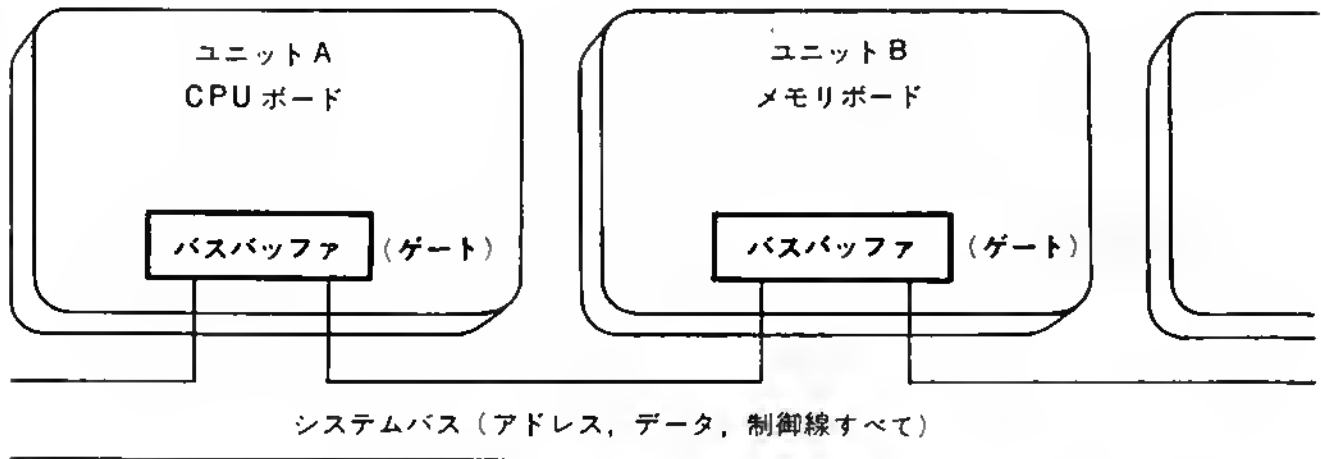
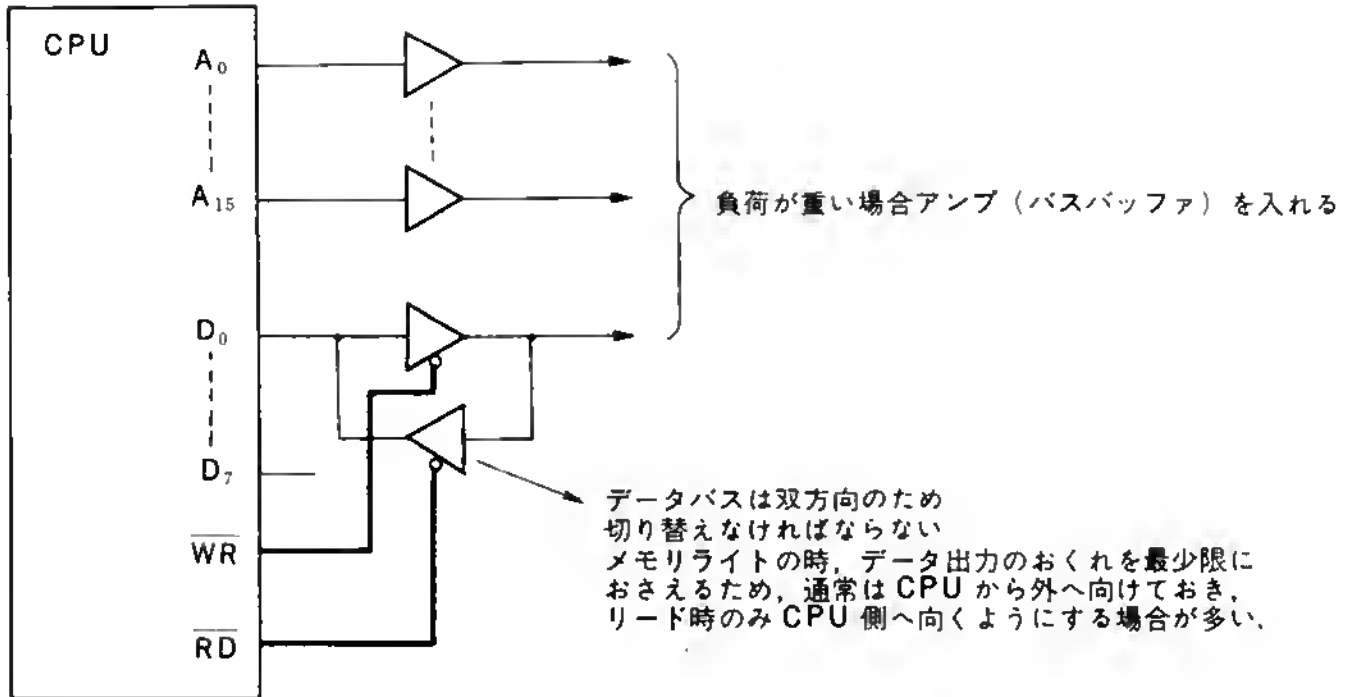


TTL のロジック IC には、ファンイン、ファンアウトという概念があります。マイコンの周辺の場合 "H" と "L" のほかに電氣的に切り離されたハイインピーダンス状態をとるスリーステートの端子が多いため単純には考えられなくなります。Z-80 ファミリの出力端子は、一つのノーマルタイプの TTL をドライブできる能力を持っています。LS タイプであれば四つまでです。これ以上の素子をドライブしなければならないときや接続ケーブルを長く引き回すときはバッファを入れなければなりません。特に長いケーブルを使用するときは波形の乱れやノイズ防止のため、バสดライバ、レシーバの能力に特別の考慮が望めます。ときにはフォトカップラやシュミットトリガが使われることもあります。

メモリなどが何枚ものボードで構成される場合、ボードの出入口のところにバスバッファを設けるのが普通です。単にバッファだけでなく、ゲートの働きもして、そのボード内のアドレスがアクセスされたときだけゲートが開くようにしておけば、CPU に対して無駄な負荷をかけることもなく、デバッグもやりやすくなります。

データバスのバッファは双方向性です。リード、ライトの信号によって方向を決定します。マルチ CPU システムとしたときや DMA を使用したときはきわめて複雑になりますので、システムのブロック化をするときには十分考慮する必要があります。

割り込みを使用するシステムでは、ペリフェラルは、CPU がメモリからフェッチしてくる命令を横からデータバスを見ていて、割り込みからの復帰を知るようになっています。したがって、データバスのバッファはフェッチサイクルではメモリから CPU へ向くと同時にメモリからペリフェラルへも送り込まなければなりません。



- 。ゲートはユニットの機能により、開け閉めする場合もあるが、常時開けておき他のゲートを制御するための情報を受けとる場合もある。

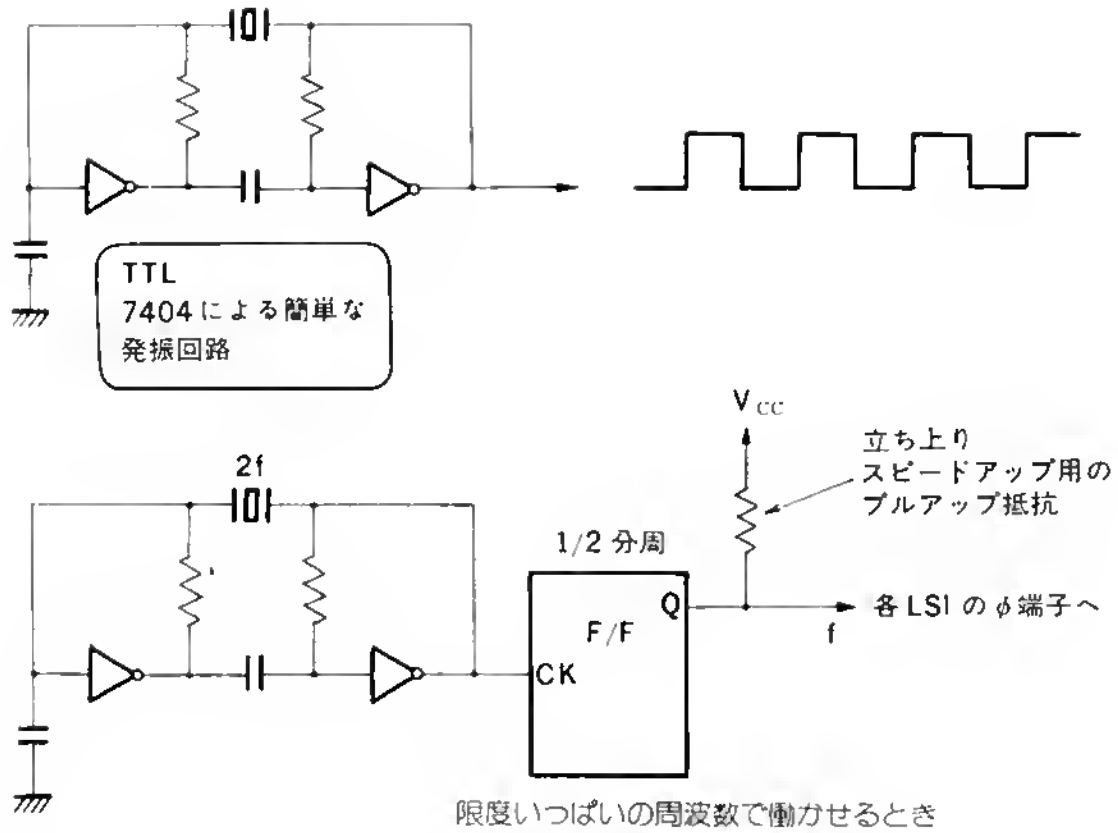
## 23 クロック

Z-80 ファミリは、ノーマルバージョンの LSI は、最高 2.5 MHz、A バージョンと呼ばれる LSI は、4 MHz までのクロックを与えることができます。システムのすべての動作はこのクロックに同期して進められますので、当然周波数が高いほど処理速度は速いわけです。しかし周辺に接続されるメモリや入出力の端末の応答速度が追従しなければ、何の意味もなく、単に CPU を待たせる回路が増えるだけになります。

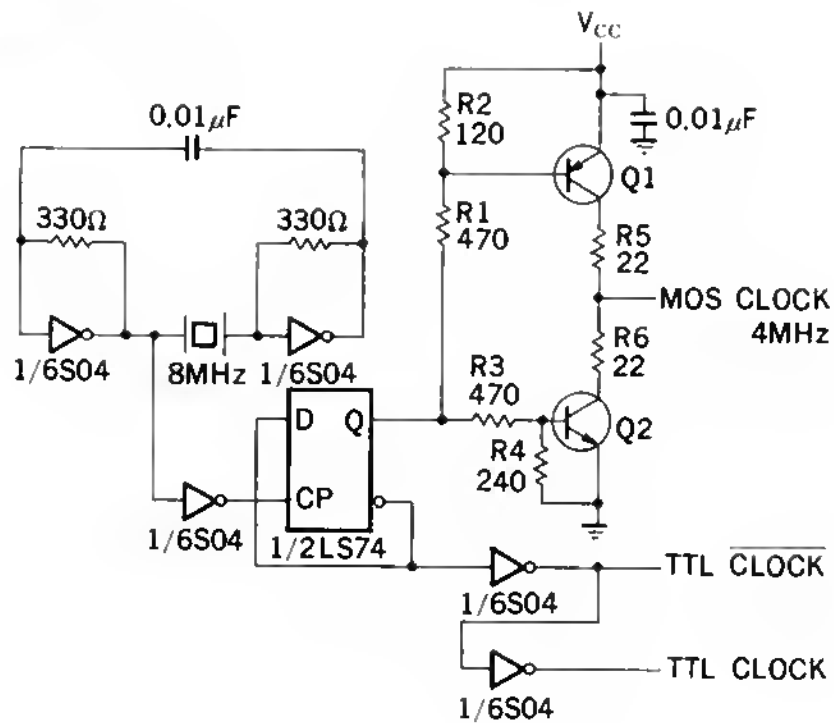
クロック(φ)は単相の方形波ですが、“H”と“L”の幅がそれぞれ 180 ns(ノーマル) 110 ns(A) 以上となっています。したがって周波数が規定以下であってもデューティ比が 1 になっていないとまずい場合があります。2 倍の発振を分周すれば正確にデューティ比 1 のパルスが得られます。

立ち上り立ち下りの時間は、最大 30 ns と規定されています。これは TTL では実現できない値ですが、かなりのオーバースペックのようです。実際には TTL の出力にプルアップ抵抗を付ける程度で実用化している例があります。

クロックパルス 1 周期を**クロックサイクル**と呼びます。マシンサイクルは、クロックサイクル数個によって一つずつ進められます。1 クロックサイクルは、(クロック周波数の逆数) 秒ですから、2.5 MHz なら 0.4 μs、4 MHz なら 0.25 μs となります。



[理想的な回路例] (サイログ社マニュアルより)



## 24 リセット

CPUのリセット (**RESET**) 端子へのリセット信号は、パワーオン時には与えなければなりません。CPU 以外へも同じ信号を与えます。PIO と DMA はパワーオン リセットの機能を内部に持っていますので必要ありませんが、与えてもよいのです。リセットの時間ば、電源電圧が安定し、クロックが安定に与えられてから最低3クロック サイクル分 (2.5 MHz クロックであれば、1.2  $\mu$ s) 以上です。実用上問題なければ、長いほど安全といえます。

CPU は、このリセット信号により

プログラム カウンタ (PC)

I レジスタ

R レジスタ

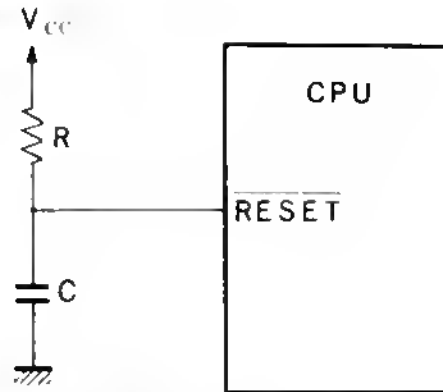
をゼロにして、割り込みモードを 0 として、割り込みを受け付けない (インタラプト ディスエーブル) 状態にします。これ以外のレジスタ (もちろん、CPU 外の RAM も) は変化しません。

リセット信号がノンアクティブ (負論理ですから "H") になると、まず、ゼロ番地の命令をフェッチすることから動作を始めます。もし I レジスタを使用するなら、ゼロ番地からのプログラムで I レジスタへ必要な数値を書き、割り込みモードを希望のモードに設定し、その他必要な初期化を行なってから、割り込みを受け付け可の状態 (インタラプト イネーブル) にする [EI] 命令を実行させます。またスタックポインタ (SP) の設定もしなければなりません。

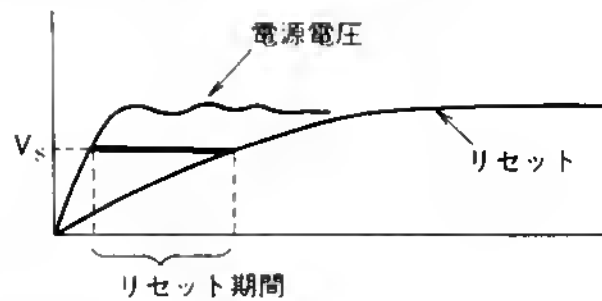
リセット信号は、パワーオン時に与えることはもちろんですが、プログラムが意図しない無限ループに入ってしまうこともあり、特にデバッグ中には、手動で電源を入切せずにリセットできるようにしておく大変便利です。ただし何らかの動作中に不用意にリセットすると困ることもありますので注意が必要です。リセット期間中はメモリリフレッシュ信号は出ません。



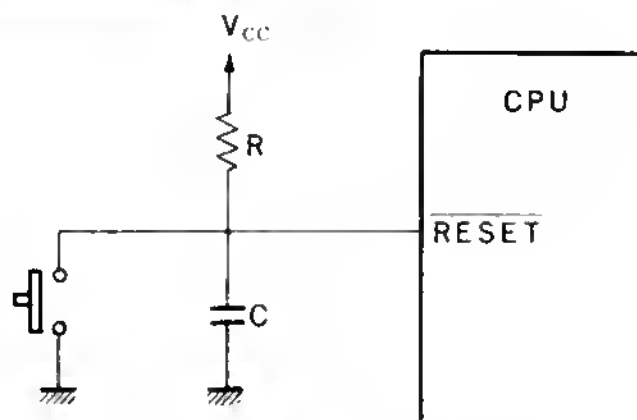
### パワーオンリセット



$R$ と $C$ により数 ms の時定数を持たせる



### マニュアルリセット



パワーオンリセットも兼ねる、スイッチのチャタリングは  $C$  により吸収される、

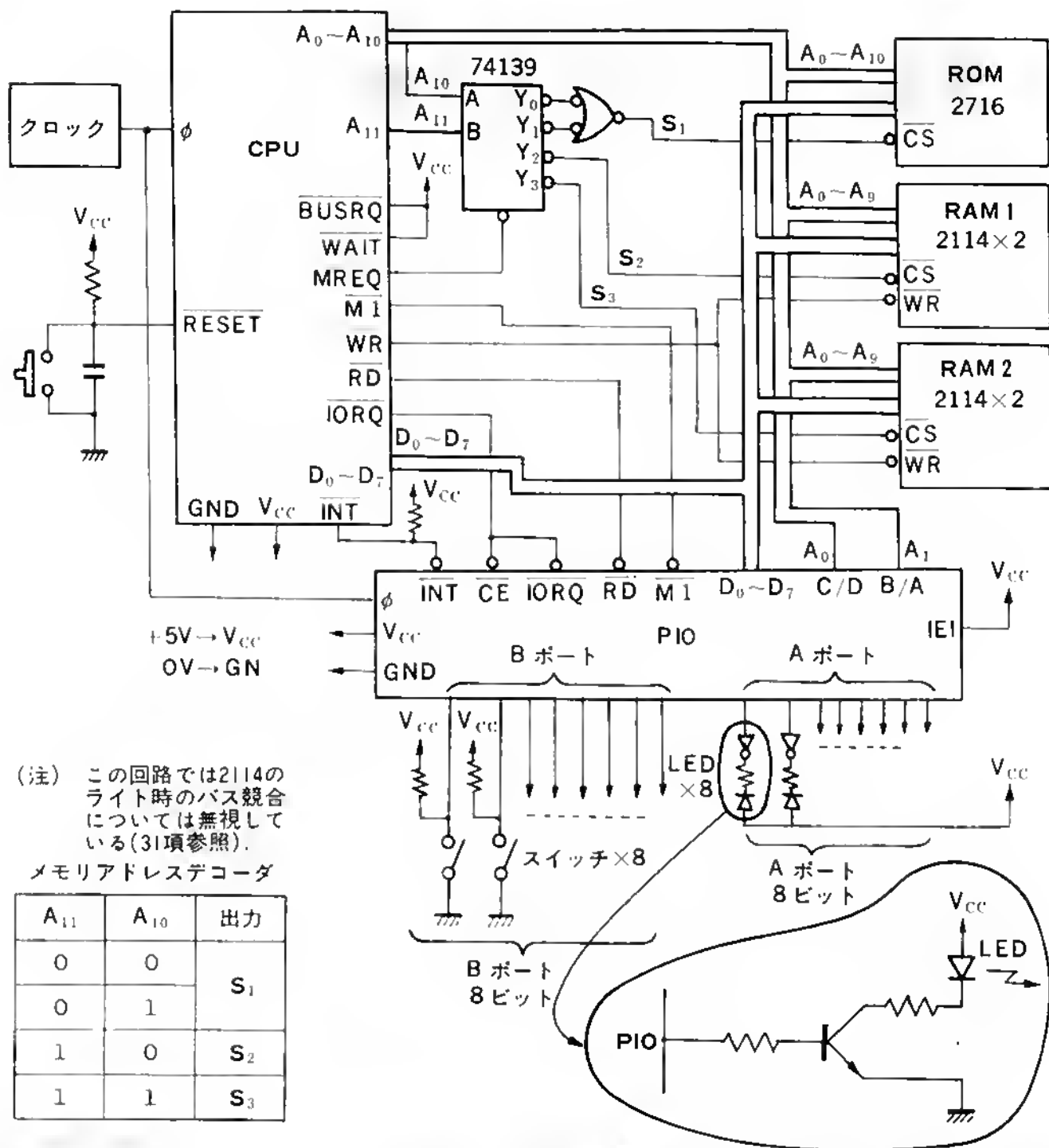
- \* リセットにノイズがのることを想定すると、シュミットトリガ回路を通した方が確実といえます。

## 2.5 システム構成

マイコンシステムの構成は、目的によって大きく異なります。普通は CPU 1 個に対して、入出力のポート数あるいはビット数を満たすだけの IO ペリフェラル、そしてプログラムを収容しきれだけの ROM と、データエリア（ワーキングエリア）としての RAM、IO とメモリのアドレスデコーダ、クロック、リセットで構成されます。さらに大きなシステムでは DMA を採用したり CTC を追加したり、場合によっては CPU を複数にしたマルチ CPU システムとすることもあります。マルチ CPU システムは、目的の仕事がはっきり区分できるときはプログラムが楽になり、効率がよくなる点でメリットがあります。

入力にスイッチ 8 個、出力に LED 8 個、プログラムは 2 K バイト以下、データエリアは 2 K バイト以下、といった簡易なコントローラの例を図に示します。メモリは合計 4 K バイトですので、アドレスバスは下位 12 ビットだけを使い、上位 4 ビットはあそばせてあります。したがって 16 進数で表現すると 3 桁だけが有効で上位桁は何であってよいわけです。何であってよいということは、何であってもしずれかのメモリがセレクトされてしまうので、これ以上増設することはできなくなります。IO ポートには PIO を 1 個使っておりますので、アドレスは四つだけ必要（詳細は PIO の項参照）になり、下位 2 ビットだけ有効です。

このシステムでもスイッチをマイクロスイッチや光センサに、LED をリレーやソレノイドに変えれば、メカニズムのコントロールロジックを置き換えたり、簡単なシーケンスコントローラとして実用化することができます。



各メモリLSIの中でデコードしてくれるので  
最小値と最大値のみを記入した

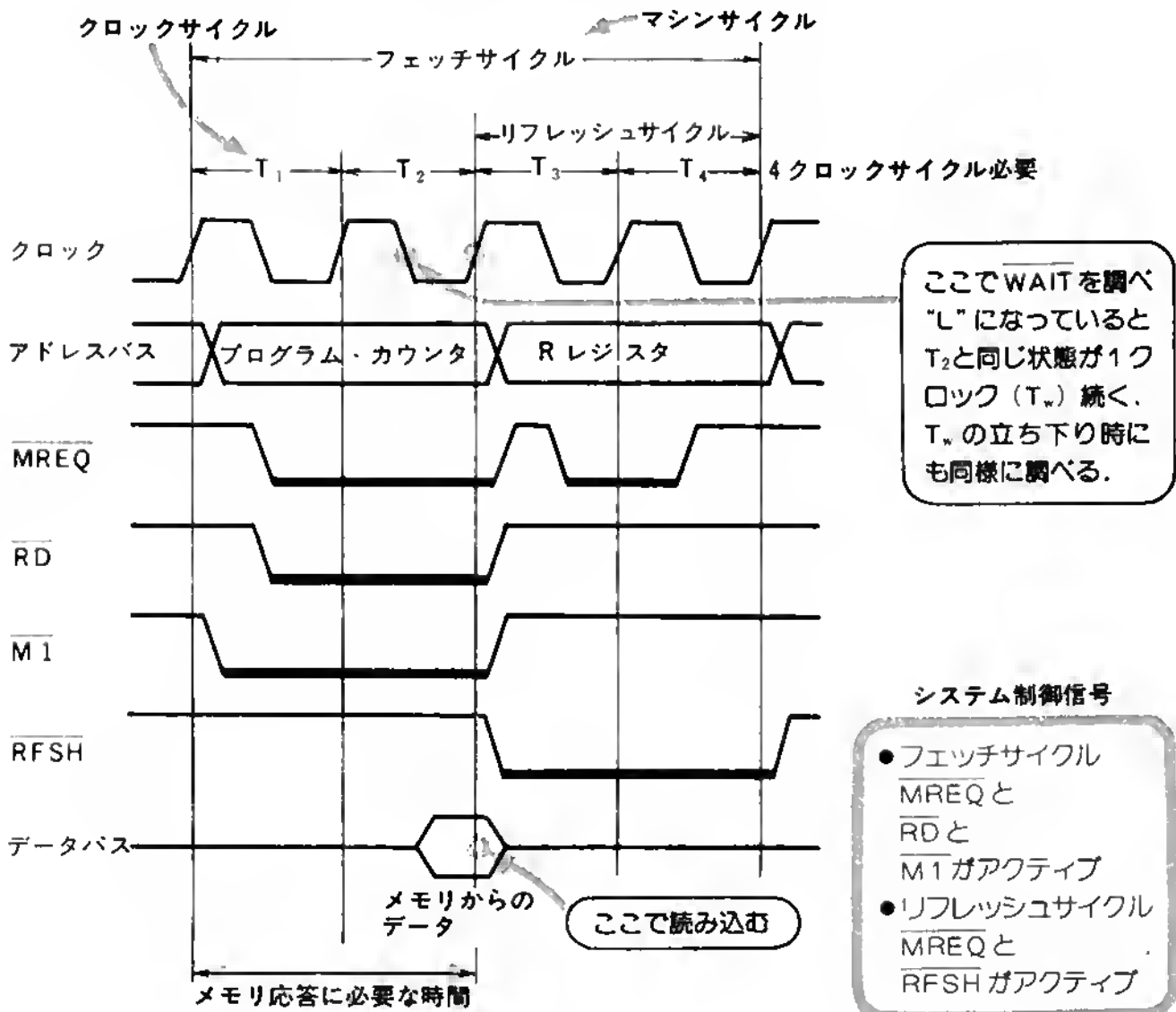
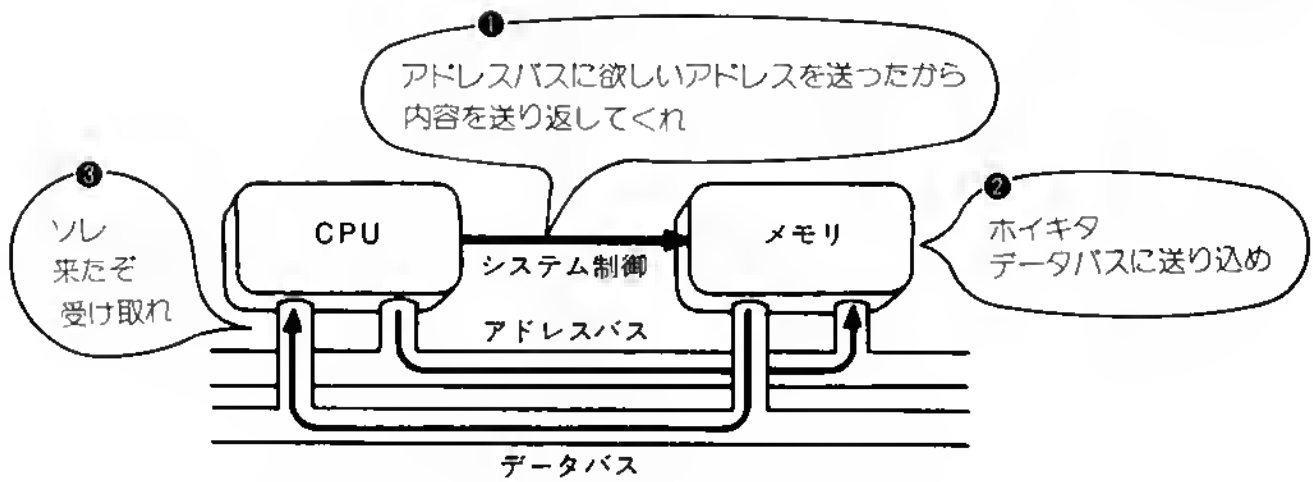
## 26 フェッチサイクルの動作

一つの命令を実行するとき、最初に CPU はフェッチサイクルに入ります。プログラムカウンタの値に示されるアドレスのメモリからオペコードを読み込み、解析します。そしてプログラムカウンタへ1を加え次の読み込みにそなえます。

フェッチサイクルが始まるとアドレスバスへプログラムカウンタの値が乗せられ、同時にエムワン( $\overline{M1}$ )信号が“L”すなわちアクティブになります。アドレスバス上のデータの安定するのを待ってメモリリクエスト ( $\overline{MREQ}$ ) 信号が、続いてリード ( $\overline{RD}$ ) 信号がアクティブになってメモリをアクセスします。3番目のクロック  $T_3$  の立ち上りで、CPU はデータバスの内容を読み込み (サンプリング) ます。 $T_2$  が立ち下るときにウェイト ( $\overline{WAIT}$ ) 信号がアクティブになっていると、そのままの状態ウェイトサイクル  $T_w$  が  $T_2$  と  $T_3$  の間に入ります。 $T_w$  の立ち下りでもウェイト ( $\overline{WAIT}$ ) 信号を見ますので次へ進めたいときは、ノンアクティブ “H” に戻さなければなりません。応答の遅いメモリを使うときに、これで時間待ちさせることができます。 $T_3$  ではダイナミック RAM をリフレッシュするためのリフレッシュアドレスがアドレスバスの下位 7 ビットに出され、リフレッシュ ( $\overline{RFSH}$ ) 信号も同時にアクティブになります。メモリリクエスト ( $\overline{MREQ}$ ) 信号もアドレスバスが安定しているであろう期間にアクティブになります。

リフレッシュ中は、CPU の内部では命令の解析が行なわれ、次のサイクルに何をすべきか判断しています。8 ビットの汎用レジスタ間の転送命令などでは 4 クロックのフェッチサイクルだけで実行を完了してしまいます。

フェッチサイクルのメモリリクエスト ( $\overline{MREQ}$ ) 信号は、データバスをサンプリングする約 1.5 クロック前に出されますので、この時間内にメモリが応答しなければなりません。しかし問題になることは少なく、万一のときもウェイト ( $\overline{WAIT}$ ) 信号で解決できます。



## 27 メモリリードサイクル

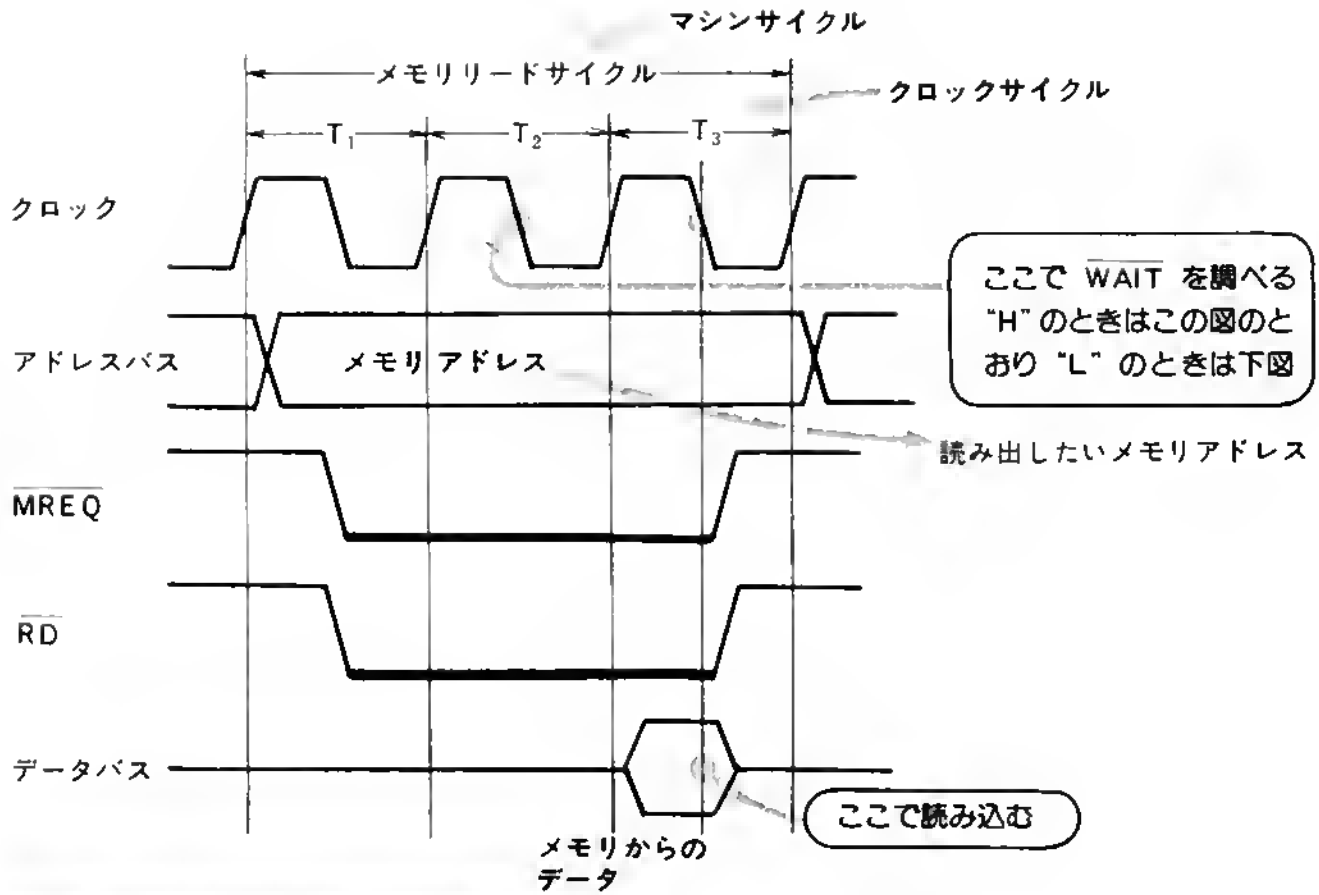
メモリからデータを読み出して、レジスタや他のメモリへ転送する命令の1サイクルとして、メモリ リード サイクルへ入る場合と、命令語のオペランドを読み出すためのメモリ リード サイクルがあります。動作はどちらも同じで、通常<sup>\*1</sup> 3クロック サイクルの間に実行されます。

アドレス バスに読み込むべきアドレスが送出され、これが安定するのを待つてメモリ リクエスト ( $\overline{\text{MREQ}}$ ) 信号とリード ( $\overline{\text{RD}}$ ) 信号が同時に出力されます。2番目のクロックは、メモリが応答し、データ バスへ情報を乗せてくるのを待つ時間です。もしメモリの応答が間に合わない場合には、立ち下りの前後にウェイト信号を入れることができます。3番目のクロックの立ち下り時点に、CPU はデータ バスのゲートを開き、内部へデータを読み込みます。

リード ( $\overline{\text{RD}}$ ) とメモリ リクエスト ( $\overline{\text{MREQ}}$ ) 信号が出てから CPU が読み込むまでの時間は、フェッチ サイクルでは 1.5 クロックでしたが、メモリ リード サイクルでは 2クロックあります。クロックを 2.5 MHz とすると、1クロック当りの時間は 400 ns<sup>\*2</sup> ですから、フェッチのときは 600 ns、リードのときは 800 ns となります。したがって、アクセス タイムが 450 ns ぐらいの標準的なメモリであれば問題なく使用できます。

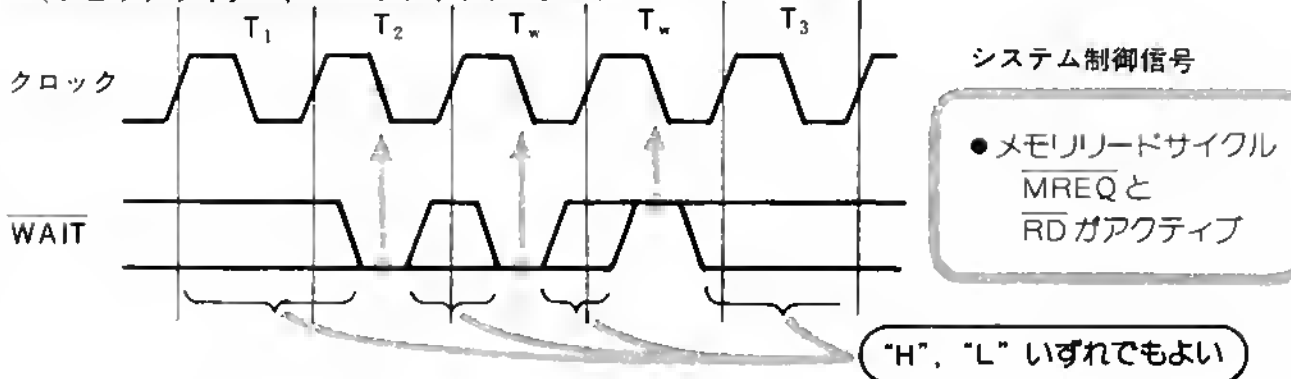
<sup>\*1</sup> 命令により例外はある。

<sup>\*2</sup> ns : ナノセックといい、 $10^{-9}$  秒。



### ウェイトサイクル ( $T_w$ ) を含む場合

(フェッチサイクル、メモリライトサイクル、IO サイクルも同じ)



$T_w$  では信号は  $T_2$  と同じ状態を保っている

## 28 メモリライトサイクル

CPU からメモリに対する書き出しのマシン サイクルです。レジスタからメモリ、メモリからメモリへの転送命令などで実行されますが、サブ ルーチン コールのコール命令で、戻り番地をスタッカへ格納するときや、割り込みがかかったときも実行されます。いずれも同じタイミングで通常\* 3クロックです。前項のリード サイクルとほとんど同じで、異なるのはリード ( $\overline{RD}$ ) 信号が出ずに、ライト信号が出る点と、CPU からデータバスへデータが乗せられる点です。

ライト ( $\overline{WR}$ ) 信号は、メモリ リクエスト ( $\overline{MREQ}$ ) 信号より1クロック遅れて出されます。これは、データ バスが安定するのを待つためで、ライト ( $\overline{WR}$ ) 信号でメモリの読み込みのゲートをコントロールすることができるようになっていきます。

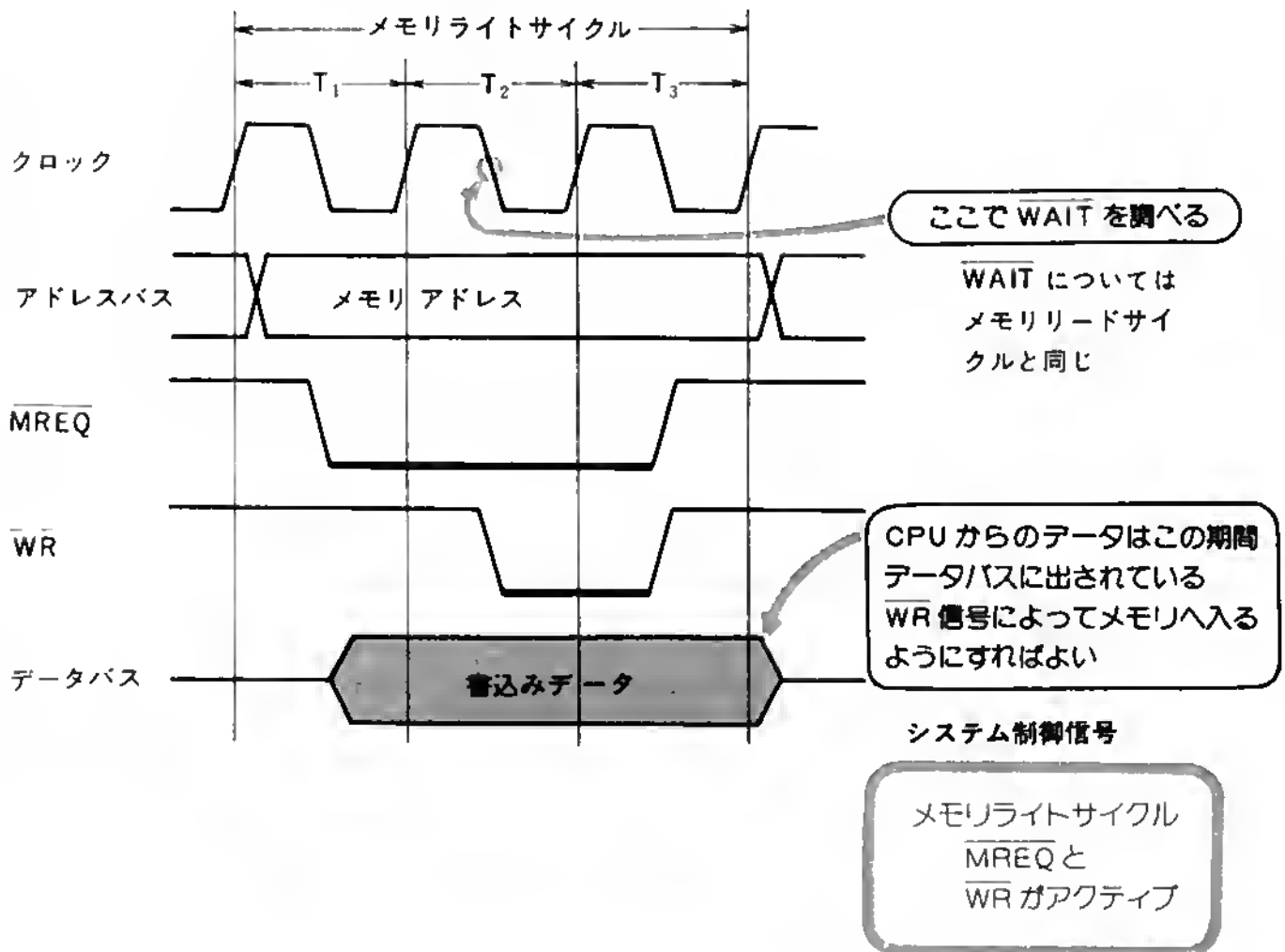
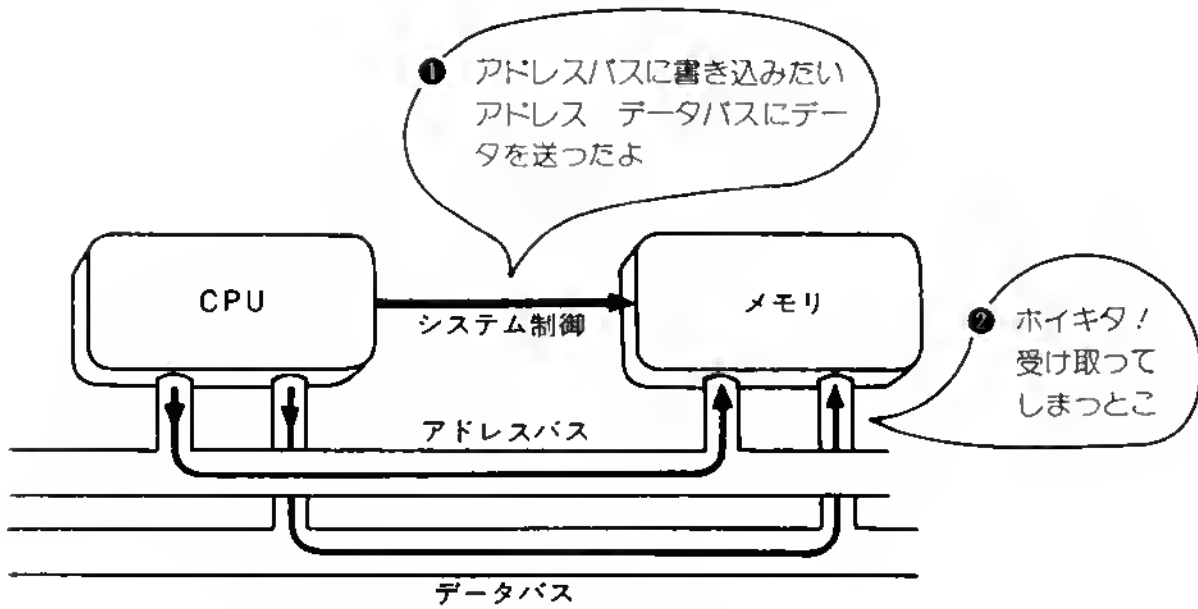
### メーカーの立場とユーザの立場

マイクロプロセッサは、一つの電子部品との見方からすると、その使い方は、ユーザの責任において研究されなければなりません。真空管やトランジスタでも、メーカーの発表するのは一使用例にすぎず、回路構成や応用製品の信頼性については、すべてユーザの研究や雑誌等のレポートを参考にして使用されてきました。

コンピュータと呼ばれることから、大型コンピュータ並の指導やサービスをメーカーに期待すると、あてがはずれます。LSI の単価には、これらの費用は含まれていないからです。マニュアルも有償である場合がほとんどです。マニュアルは、指導書ではありませんので、解りにくいものです。基本的に「何をどうするとどうなります」という LSI の機能を述べ連らねてあるにすぎません。書物や雑誌に目を向け、いろいろな事例に接することが大切です。

\* 命令により例外はある。





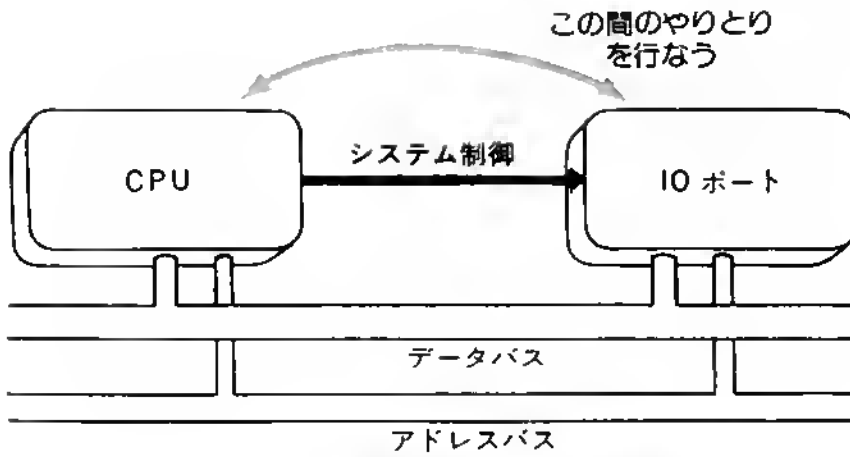
## 29 IOリードサイクル、IOライトサイクル

IOポートに対するリード、ライトは、アドレスバスの下位8ビットが有効なアドレスとして使われます。上位8ビットへは何らかの状態が出力されますので、無視しなければなりません。

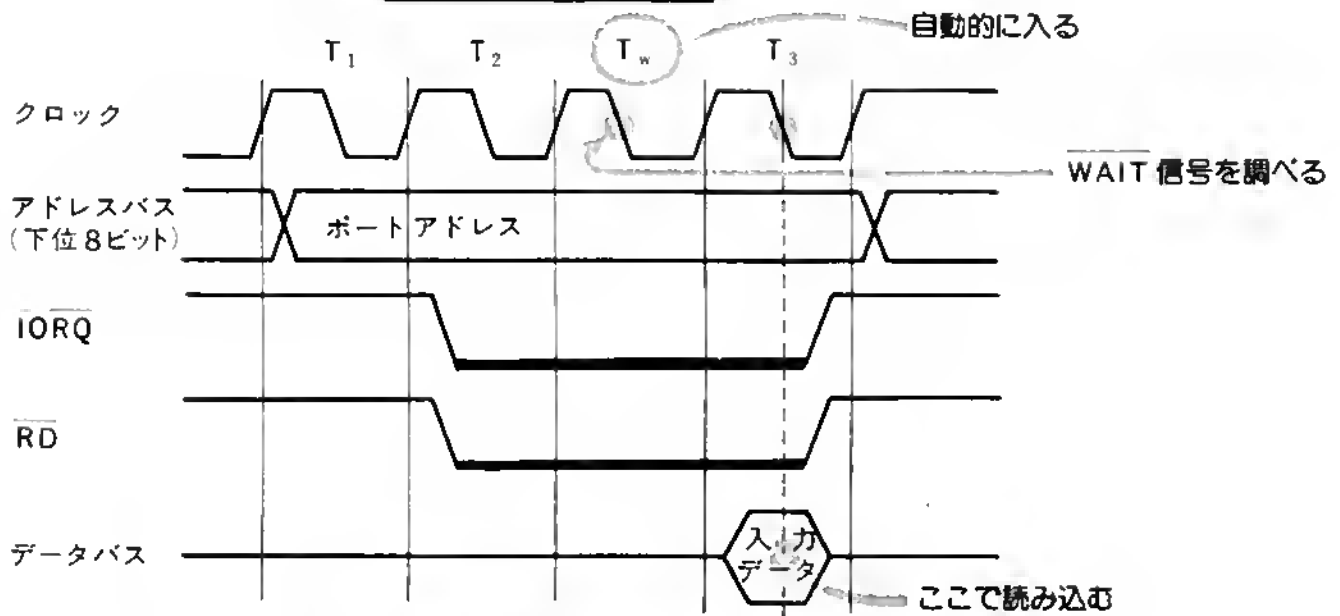
イン [IN] 命令、アウト [OUT] 命令と、これらの繰り返しの命令を実行したときに、このサイクルへ入ります。

IOサイクルでは、メモリの読み書きより多く時間を必要とすることが想定されますので、ウェイトサイクル  $T_w$  が自動的に入ります。もっと長く時間待ちさせたいときは、この  $T_w$  の立ち下りの前後にウェイト ( $\overline{\text{WAIT}}$ ) 信号を入れることになります。またリード ( $\overline{\text{RD}}$ )、ライト ( $\overline{\text{WR}}$ ) と IO リクエスト ( $\overline{\text{IORQ}}$ ) 各信号は、立ち下り、立ち上り共同じタイミングです。

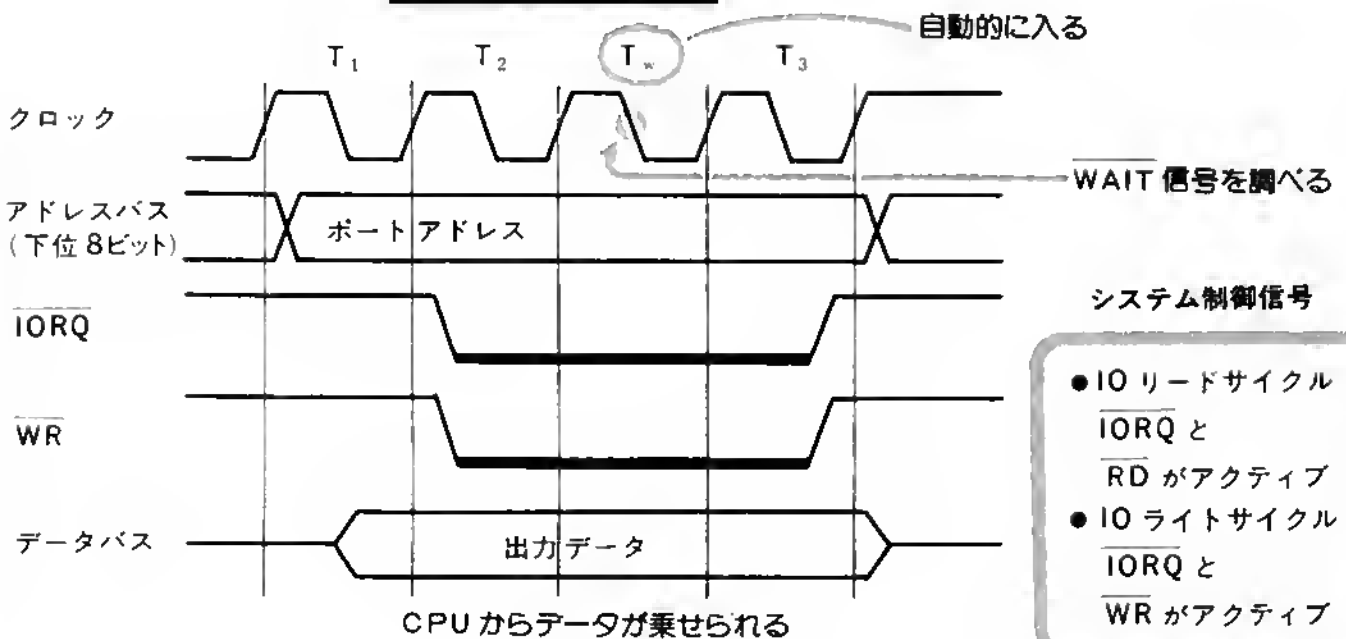
イン [IN] 命令、アウト [OUT] 命令は、入出力するポートアドレスを指定しなければなりません。直接指定の場合は A レジスタとの間でのやりとりになり、ポートアドレスを  $n$  とすると [IN A, (n)], [OUT (n), A] と書きます。間接指定の場合は、ポートアドレスを C レジスタに入れておき、[IN r, (C)], [OUT (C), r] (ここで  $r$  は A, B, C, D, E, H, L のいずれかのレジスタ) と書きます。このほかにメモリの一連の複数バイトのブロックを入出力するブロック入出力命令があります。



### IO リードサイクル



### IO ライトサイクル



システム制御信号

- IO リードサイクル  
IORQ と RD がアクティブ
- IO ライトサイクル  
IORQ と WR がアクティブ

## 30 リフレッシュサイクル

リフレッシュサイクルは、各フェッチサイクルの後半に行なわれます。アドレスバスの下位7ビットにはRレジスタの内容が出力されます。Rレジスタは1回ごとに1ずつ増やされ、ダイナミックRAMの1列ごとに再書き込みを行ないます。

このときはリード( $\overline{RD}$ )、ライト( $\overline{WR}$ )の信号の代わりにリフレッシュ( $\overline{RFSH}$ )信号が出て、リフレッシュサイクルであることを知らせます。

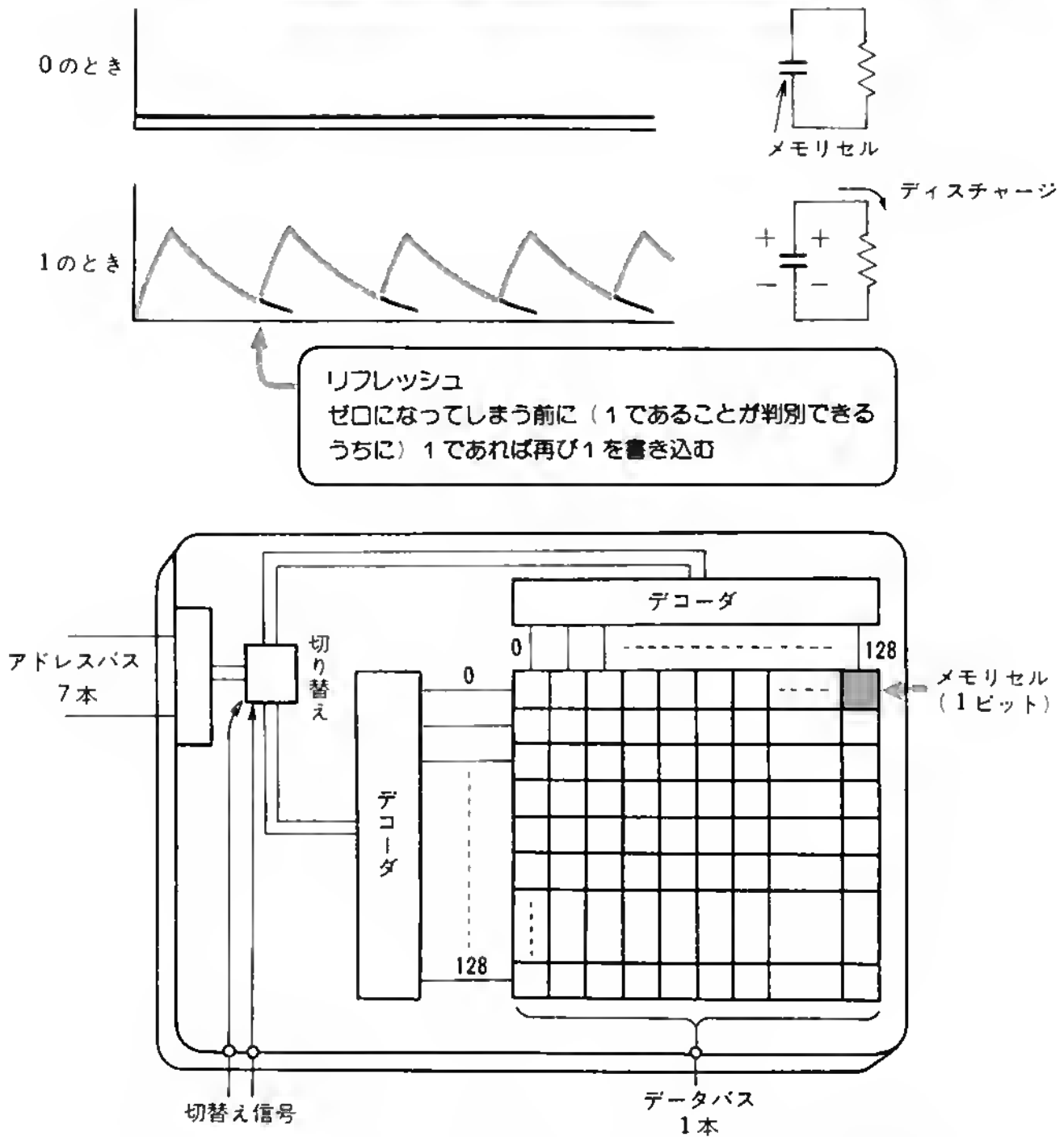
リフレッシュは長くても2ms以内に1回以上行なわなければならないので、Z-80を最高速で働かせたときでも16KビットのダイナミックRAMが限界になります。16KビットのダイナミックRAMの代表的なものに4116と呼ばれるものがありますが、 $16K = 2^{14}$ で、アドレスバスは14ビット必要です。ところが上位、下位7ビットずつを時分割して与えますので、アドレス入力は7端子しかありません。リフレッシュの場合は、7ビットのアドレスだけを順次与えればよいのです。したがってRレジスタは7ビットとなっています。

16Kビット以上のダイナミックRAMのリフレッシュは、Z-80の機能とは別にリフレッシュ回路を作らなければなりません。

### MOS LSIの取扱い

Z-80ファミリに限らず、MOSプロセスのLSIは、端子のインピーダンスが高いため帯電しやすく、チップの絶縁が破壊することがあります。保存するときはモスパッドと呼ばれる導電性のスポンジにさしてピン間をショートしておけばまちがいありません。また、ピンに直接手で触れたり、帯電していそうな所へさわらないようにしなければなりません。ハンダゴテも絶縁のよいものか、アースのとれるものを使用します。実際には、そう簡単にこわれてしまうわけではありませんが、念のために心がけたほうがよいでしょう。

## ダイナミック RAM の模式図 (4116)



- アドレスバス 7 本へ 上位 7 ビット, 下位 7 ビットを時分割で与える.
- いま, 上位が与えられているか下位が与えられているかを切替え信号で知らせる.
- リフレッシュの場合は, 一方だけを与え, 他方は全メモリセルを選択する. → データ出力は無意味 (デタラメ) → データ出力なし.

## 3.1 CPUと周辺の接続（インターフェース）

各マシンサイクルごとにアドレスバスとデータバスの意味が変わりますので、どの動作をしているかによって、メモリとIOポートのどれを選ぶか、また、書き込みか読み出しかの切り替えをしなくてはなりません。

まず、アドレスバスをデコードして、セレクト信号を作ります。次にメモリリクエスト（ $\overline{\text{MREQ}}$ ）とIOリクエスト（ $\overline{\text{IORQ}}$ ）信号によりメモリとIOを切り替えます。ここでIOリクエスト信号はエムワン（ $\overline{\text{M1}}$ ）信号と同時にアクティブになったときは別の意味（割り込み応答）になりますので、このときはIOをセレクトしてはいけません。Z-80ファミリではセレクトしても問題ありません。

IOポートにZ-80ファミリのLSIを使うときは、同じ名称のピン同士を接続すれば、システム制御信号に関しては何ら心配はいりません。2個以上のLSIを接続するならば、アドレスデコードしたチップセレクト信号を使わなければなりません。

メモリは種類によって異なります。ROMの場合は、チップセレクト信号は、リード信号が出たときだけ与えるようにしなければなりません。RAMの中にはアウトプットドライブ（ $\overline{\text{OD}}$ ）端子があるものがありますが、この端子へはリード信号を直接与えます。ライトイネーブル（ $\overline{\text{WE}}$ ）端子へはライト信号を与えればよいのです。アウトプットドライブ端子がないRAMの場合、ライトイネーブル端子へ与える信号には工夫がいります。メモリライトサイクルでは、メモリリクエスト信号が出て少したってからライト信号が出ますが、このライト信号が出るまでの時間、メモリはリード動作をしてしまい、CPUからのデータとメモリからのデータがデータバス上でぶつかってしまいます。実際問題としてはLSIをこわしてしまうほどのことはないようですが、さけたほうが賢明でしょう。リード信号とリフレッシュ信号がなくかつ、メモリリクエスト信号が出たときだけライトイネーブル端子へ与える信号を作れば解決できます。

システム制御信号

	M1	MREQ	IORQ	RD	WR	RFSH	
フェッチサイクル							メモリ
メモリ リード							
メモリ ライト							
IO リード							IO ポート
IO ライト							
リフレッシュ							ダイナミックメモリ
割り込み応答							IO ポート
		メモリ	IOポート	データの方向			
		アドレス空間					

- この信号の組み合わせでメモリやIOポートをイネーブル（活かす）したり、ディスエーブル（殺す）したりする（ように接続する）。  
アドレスデコーダをコントロールしてセレクト信号が出るようにしたり、止めたりしてもよい。
- Z-80 ファミリーで構成するときは、同じ信号線どうしを接続すればよい。

## 3.2 ウェイト信号とホルト信号

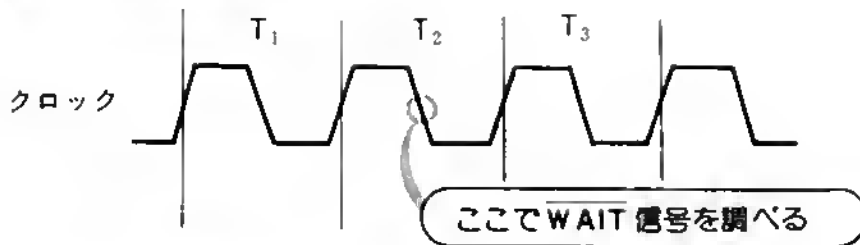
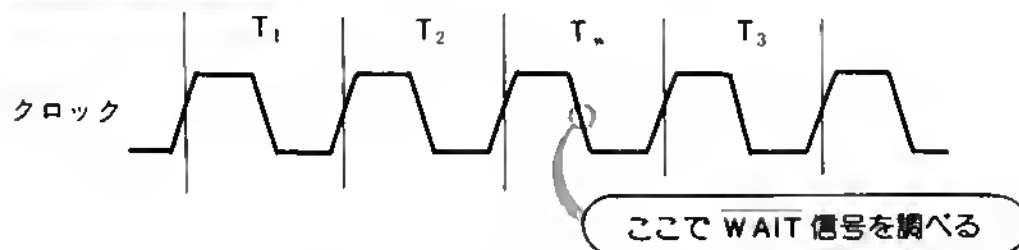
**ウェイト (WAIT) 信号**は、メモリやIOの応答速度がCPUのタイミングに間に合わないときに、ダミーのクロックサイクルを入れて、CPUを待たせるときに与えます。CPUは各マシンサイクルの2番目のクロックの立ち下りの時点で、ウェイト信号端子の状態を検知して、このとき“L”になっていれば、次のクロックをウェイトサイクルとして何もしないで待ちます。このウェイトサイクルの立ち下りのときも同じく検知していますので、“H”になるまで空転していることになります。ウェイト中はリフレッシュは行なわれません。

IOライトサイクルとIOリードサイクルでは一つ、割り込み応答のフェッチサイクルでは二つのウェイトサイクルが自動的に挿入されます。

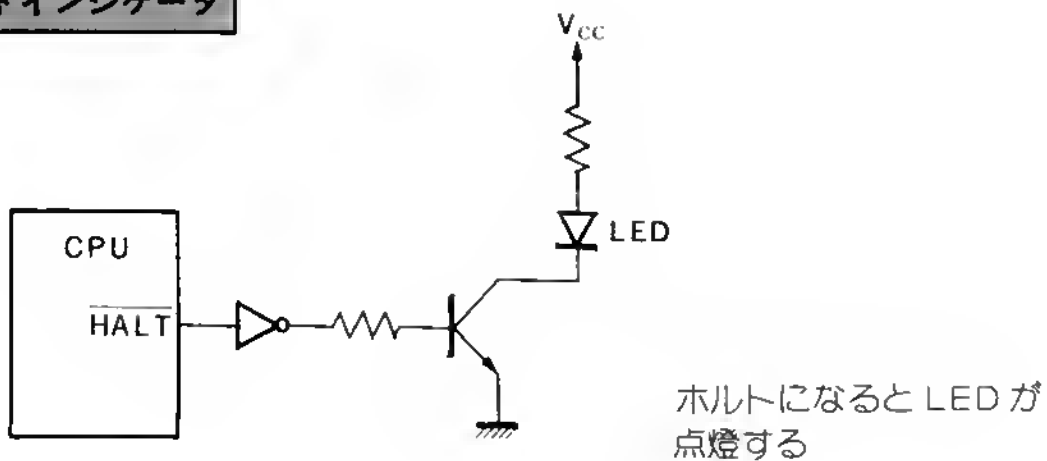
簡単なシステムではウェイトの端子を使用しないことのほうが多いと考えられますが、そのときは $V_{CC}$ へ続いて“H”レベルにしておかないと、CPUは動作したり止まったり、不安定な状態になってしまいます。

**ホルト (HALT) 信号**は、CPUがホルト [HALT] 命令を実行したときに出される信号です。[HALT] 命令を実行すると、プログラムカウンタの進行を止めてしまいますので、CPUが停止したのと同じです。普通のプログラムでは、このようなことは起こり得ませんので、異常があったことを外部へ知らせる信号として使えます。ホルト状態からの解除は、リセット信号を与えるか、割り込み受け付け可になっているときは、割り込みをかければよいのです。ホルト状態では、ノンオペレーション [NOP] 命令といって、フェッチサイクルだけで何もしない命令をくり返し実行しています。このようにしてある理由は、ホルト中もメモリリフレッシュを絶やさないためです。



**$T_w$  がない場合** **$T_w$  がある場合**

- $T_2$  の立ち下りまたは  $T_w$  の立ち下りで  $\overline{\text{WAIT}}$  を調べ、“H” ならそのまま、“L” なら  $T_w$  を挿入する。
- 挿入された  $T_w$  の立ち下りでも  $\overline{\text{WAIT}}$  を調べ、“H” なら次は  $T_3$ ，“L” ならもうひとつ  $T_w$  を挿入する。
- $\overline{\text{WAIT}}$  が “H” になるまで何回でも  $T_w$  が入る。

**ホルトインジケータ**

## 3.3 割り込みの概念

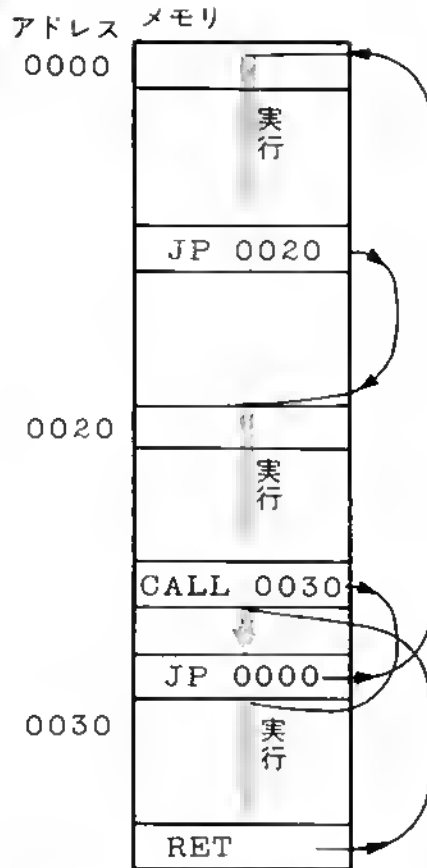
CPUは、リセット信号( $\overline{\text{RESET}}$ )が入ると、ゼロ番地から順次命令を実行します。命令の中に、「ある番地へジャンプせよ」という命令、ジャンプ [JP] 命令やコール [CALL] 命令があると、その番地の命令を実行します。つまり、通常はプログラムカウンタを一つずつ繰上げていき、特別の命令があると、プログラムカウンタの内容をその命令に従って変更します。

いま、このようにして処理を進めているときに、急に他の処理をしなければならないなくなったことを考えてみます。停電になってとりあえずバッテリーに切り替えるとか、端末装置から故障を知らせる信号が入ったときに、通常のプログラムの進行を止めて、対策処理をしなければならないことがあります。このようなときに、インタラプト ( $\overline{\text{INT}}$ ) かノンマスカブル インタラプト信号 ( $\overline{\text{NMI}}$ ) を入れることによって、プログラム上の命令とは関係なく、特定の番地へジャンプさせることができます。これを**割り込み**と呼びます。

割り込み機能をもっと積極的に使うことにより、処理効率を高めたり、プログラムを簡単にしたりすることもあります。たとえば、CPUからプリンタにデータを打出すよう命令を与えてからプリンタが動作を終るまでの時間は、CPUの処理速度に対して非常に長いのですが、この間ただ待つのではなく、CPUは別の仕事をしていて、プリンタからの動作終了割り込みで次のデータを打出すことができます。一度に二つの仕事をしているようにみえるのでマルチタスクとかマルチジョブと呼んで、入出力の多い仕事ではCPUの処理時間は無視され、端末装置の動作時間だけで処理速度がほぼ決まってきます。

割り込みはCPUの動作に同期せずに、外部からのハード的な要求で呼び出されるサブルーチンコールと考えてよいでしょう。

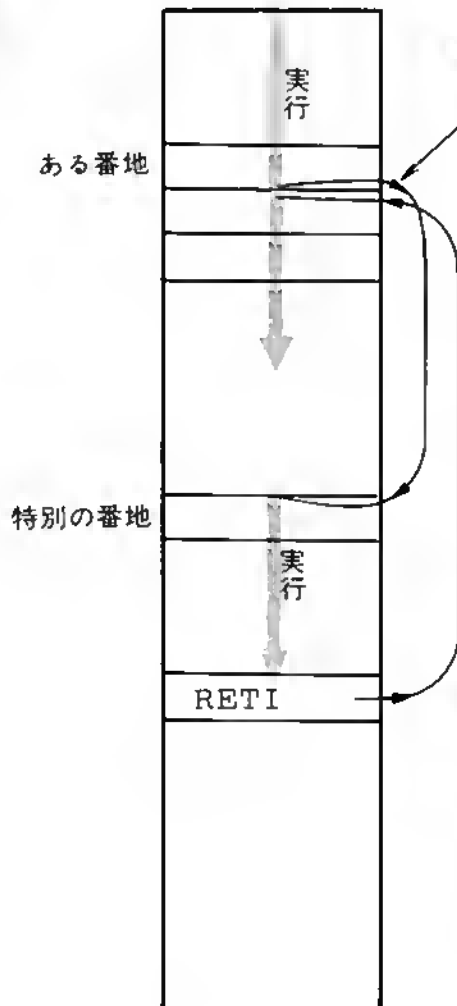
## 通常の実行



- 普通は順序よく、
- ジャンプ、コールなどの命令があれば指定の番地へ実行を移し、そこから実行する。

命令により実行アドレスを変える

## 割り込み



電気信号 (NMI か INT 端子) 割り込み信号

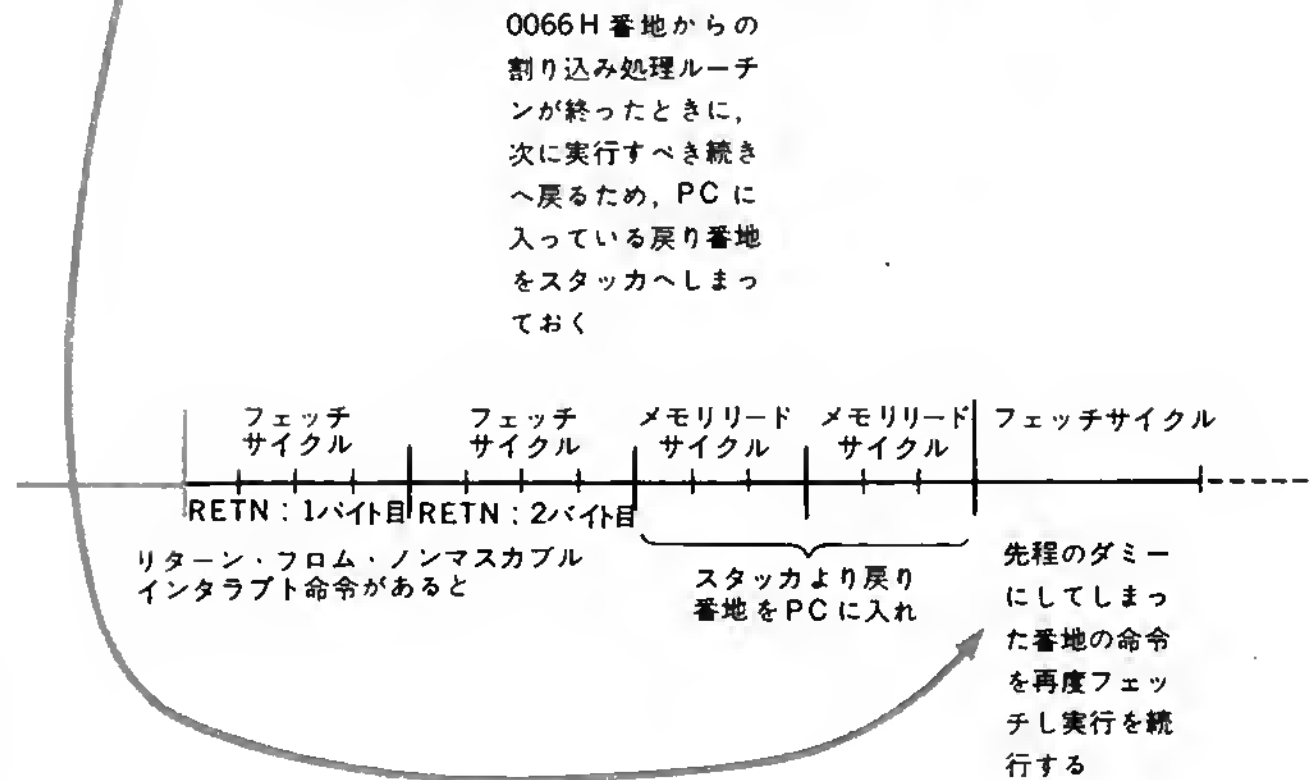
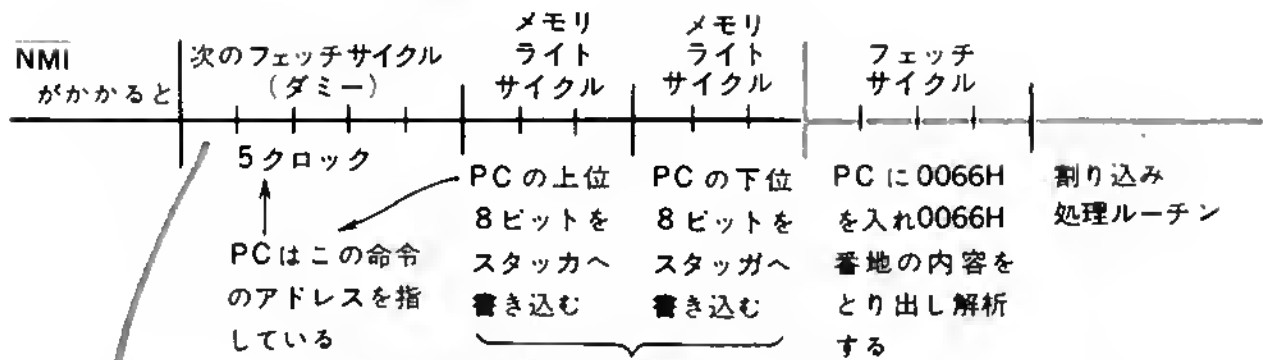
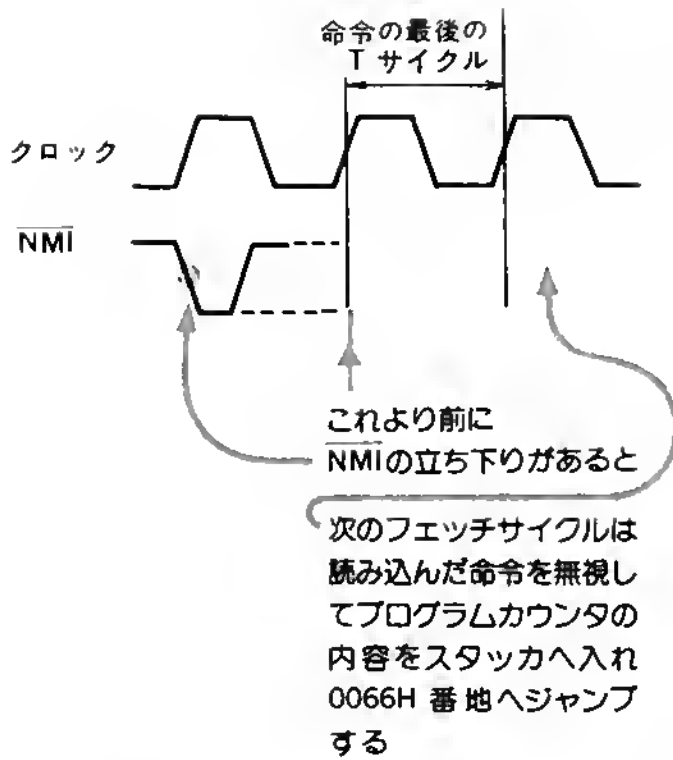
- ある番地の命令を実行中 (命令は何でもよい) 電気信号が入ると特別の番地へ実行を移す。RETI (リターン命令) があると元の命令の次 (さっきのつづき) へ実行をもどす。

電気信号により実行アドレスを変える

## 3.4 ノンマスカブルインタラプト (NMI)

Z-80 には、ノンマスカブル インタラプト ( $\overline{\text{NMI}}$ ) と、ただのインタラプト ( $\overline{\text{INT}}$ ) の 2 系統の割り込みがあります。ノンマスカブル インタラプトは、いかなるときでも受け付ける最優先の割り込みです (ただし、バス アクノリッジ中は受け付けません)。したがって、停電などシステムやオペレータにダメージを与えるような非常時の対策に使うことが多いようです。

ノンマスカブル インタラプトは名前のとおり、プログラムによってマスク、すなわち無効にすることができない割り込みです。CPU はノンマスカブル インタラプト信号線 ( $\overline{\text{NMI}}$ ) が立ち下ったとき、実行中の命令が終り次第、割り込み処理に入ります。割り込みの処理プログラムが終ったときまた元の番地へ戻らなければならないために、現在のプログラム カウンタ (PC) の内容をスタッカへ格納します。次に、プログラム カウンタへ 0066H を書き込み、新しいフェッチ サイクルへ入ります。したがって、0066H 番地をコールすることになります。割り込みがかかった時点で実行中の命令は終りまで実行し、次の命令のフェッチ サイクルもあたかも平常のとおり to 実行されますが、データバスを無視して [CALL 0066H] をフェッチしたかのように動作を続けていると考えられます (ただし、タイミングは異なります)。0066H 番地には割り込み処理プログラムを書いておかねばなりません。そして処理の最後にリターン フロム ノンマスカブル インタラプト [RETN] 命令があれば、スタッカへ格納しておいた戻り番地をとり出して、割り込みがかかったときに実行していた次の命令へ復帰します。 $\overline{\text{NMI}}$  が入ってから [RETN] が実行されるまではインタラプト ( $\overline{\text{INT}}$ ) は無視され、待たされることになります。



## 3.5 インタラプト (INT)

リセット信号が入って CPU が初期状態から実行し始めると、インタラプト信号に対するマスクは、ディスエーブルすなわち割り込みを無視するようになっています。また割り込みモードは 0 になっています。もしモード 1 か 2 で使用するのなら、[IM1] か [IM2] の命令を実行させておく必要があります。モード 2 の場合は、I レジスタとペリフェラル内のレジスタに対し数値を設定しなければなりません。これらの準備が終ったところで、割り込みを許可する命令、イネーブル インタラプト [EI] を実行させますと、これ以降割り込みがかかるようになります。割り込みを禁止するときは、ディスエーブル インタラプト [DI] です。

割り込みがかかると、次の割り込みがかからないようにディスエーブル インタラプト [DI] 命令を自動的に実行し、おのこの番地へジャンプします。割り込み処理プログラムの終りには、リターン [RET] 命令が書かれていれば、CPU は元の処理を続行するべく戻り番地へジャンプします。ただしこのとき、[RET] 命令の直前に [EI] 命令を置かなければ、割り込みは禁止されたままになってしまいます。[EI] はただちに有効になるのではなく、次の 1 命令を実行し終ったときから有効になりますので、[RET] が終了した後に割り込みを受け付けるようになります。

なお、ノンマスカブル インタラプトは信号の立ち下りエッジでかかりますが、インタラプトのほうは各命令の最後のクロック サイクルの立ち上り時点で信号の状態を検査しますので、このとき "L" になっていなければなりません。

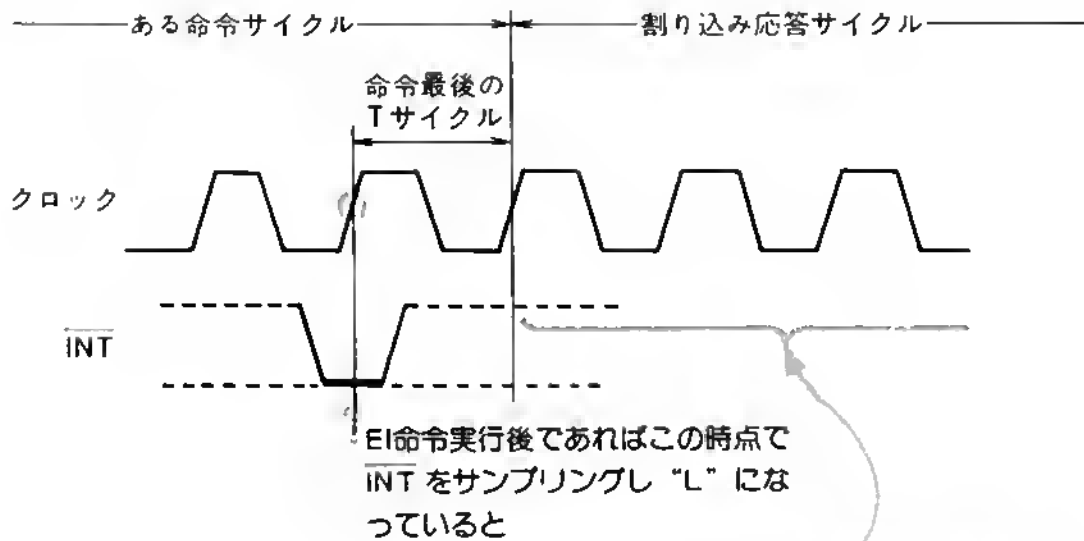
Z-80 ファミリのペリフェラルは、優先順位を決定する機能を個々に持っており、外部に回路を必要としません。この機能を使うときは、割り込み処理が終ったことを通常のサブ ルーチンからのリターンとは区別してペリフェラルに与えなければならないため、[RET] ではなくリターン フロム インタラプト [RETI] 命令を使用します。

IM 0 }  
 IM 1 } 割り込みモードを設定する命令. いずれかのモードを選びプログラ  
 IM 2 } ムの最初の方で1回実行させる.

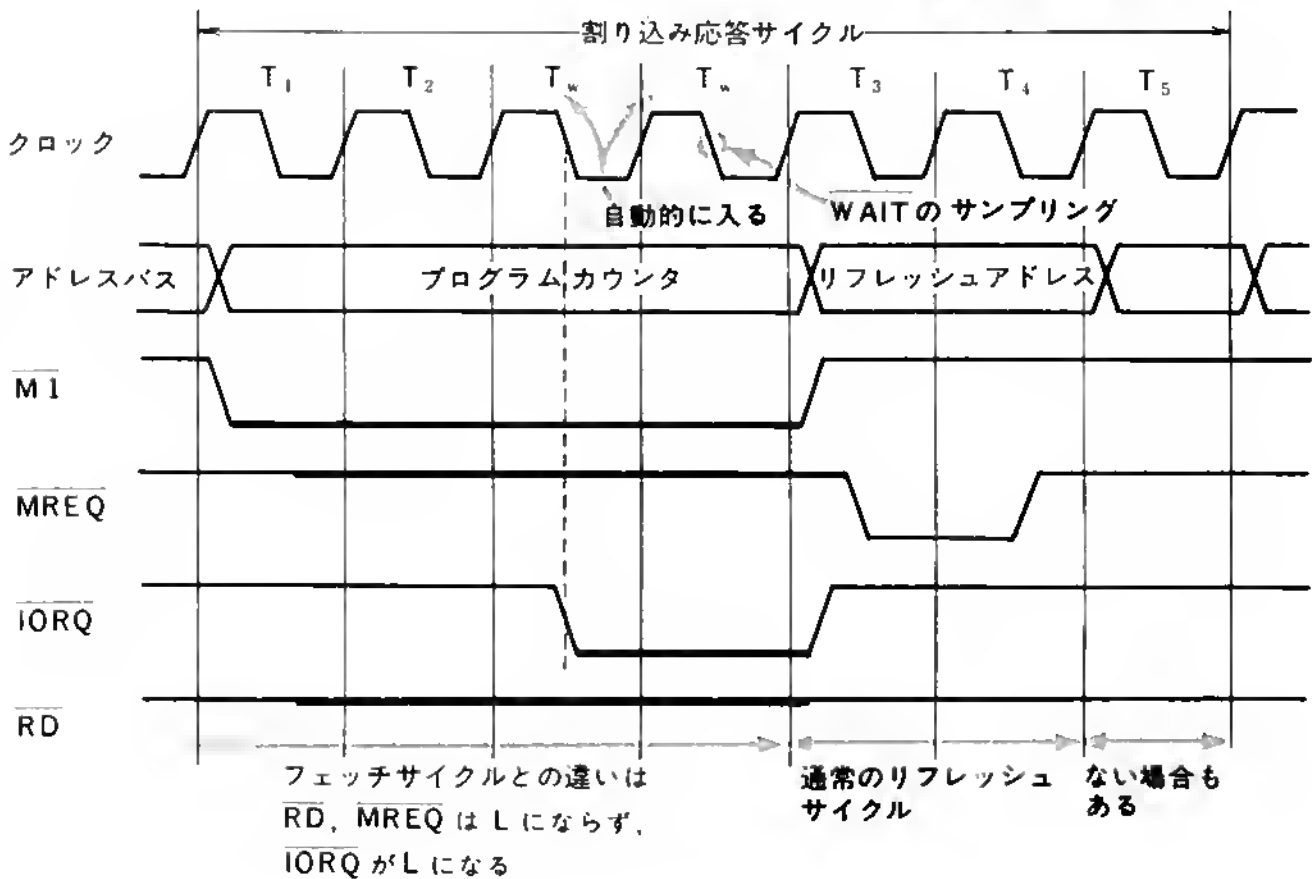
EI 割り込み信号を許可する命令

DI CPU 側で割り込み信号を受け付けなくする命令

EI を実行後 CPU は割り込みを受け付け, DI を実行すると受け付けなくなる.



次のフェッチサイクルは割り込み応答サイクルになる

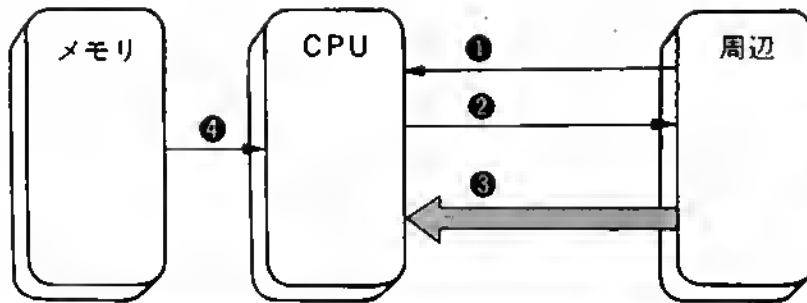


## 36 モード0のインタラプト

割り込みモード0は、8080Aと同じ動作をします。命令の最後のクロックサイクルの立ち上りのとき、インタラプト ( $\overline{\text{INT}}$ ) 信号が“L”になっていると、次のフェッチサイクルは割り込み応答サイクルとなります。フェッチサイクルとの違いは、メモリリクエスト ( $\overline{\text{MREQ}}$ ) 信号が出ずにIOリクエスト ( $\overline{\text{IORQ}}$ ) 信号が出力されることと、3、4番目のクロックサイクルにウェイトサイクルが2個、自動的に挿入されていることです。したがって、ペリフェラル側では、エムワン ( $\overline{\text{M1}}$ ) 信号とIOリクエスト ( $\overline{\text{IORQ}}$ ) 信号が共にアクティブ、すなわち“L”になったことで、割り込みがかかったことを知ります。割り込み応答サイクルでは、アドレスバスには次の命令のアドレス、すなわちプログラムカウンタの値が出ますが、外部からはこれとは関係なく、何らかの命令をデータバスに乗せてやります。この命令は主にリスタート[RST]命令か、コール[CALL]命令を使います。[RST]命令は1バイト構成ですからこれでよいのですが、[CALL]命令のときは3バイト構成ですので、あと2回メモリリードサイクルに合わせて、データバスへ割り込み原因に応じたプログラムルーチンを実行させるための命令の続き（コール命令のジャンプ先番地を意味するオペランド）を与えなければなりません。

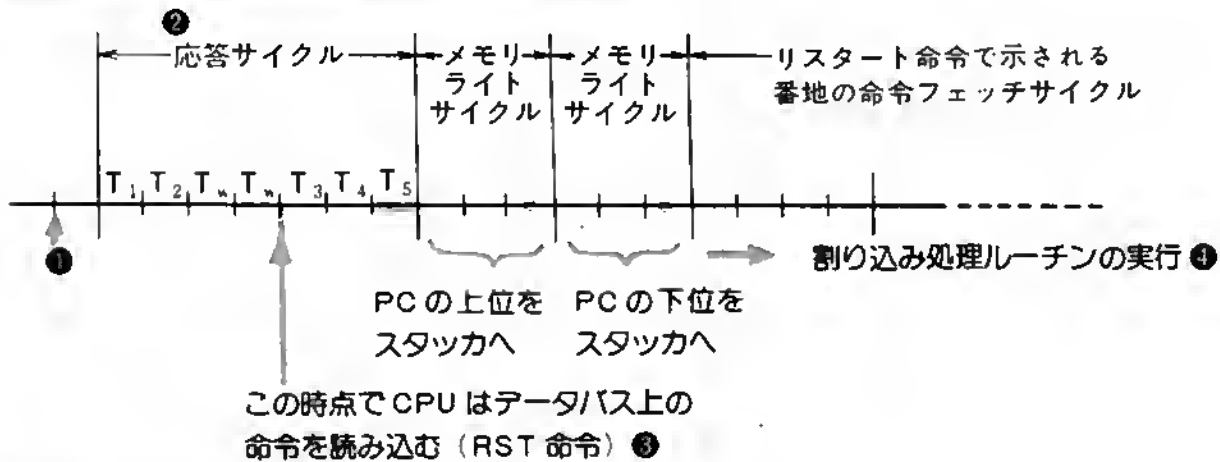
CPUの読み込みタイミングは、フェッチサイクルと同じく $T_3$ の立ち上りです。またリスタート命令を与えたときは、スタックポインタの変更をするためのクロックサイクル $T_5$ が付きますが、コール命令ではこれがなく、マシンサイクルの3番目(M3サイクル)が、メモリリードサイクルの3クロックの後に $T_4$ として続き、4クロックサイクルとなります。これは通常の場合にフェッチサイクルでコール命令やリスタート命令をフェッチしたのと同じです。すなわち応答サイクルで、アドレスバスを無視して強制的にデータバスへ命令を与えれば、それ以後は通常その命令を実行するのと同じ動作をしています。



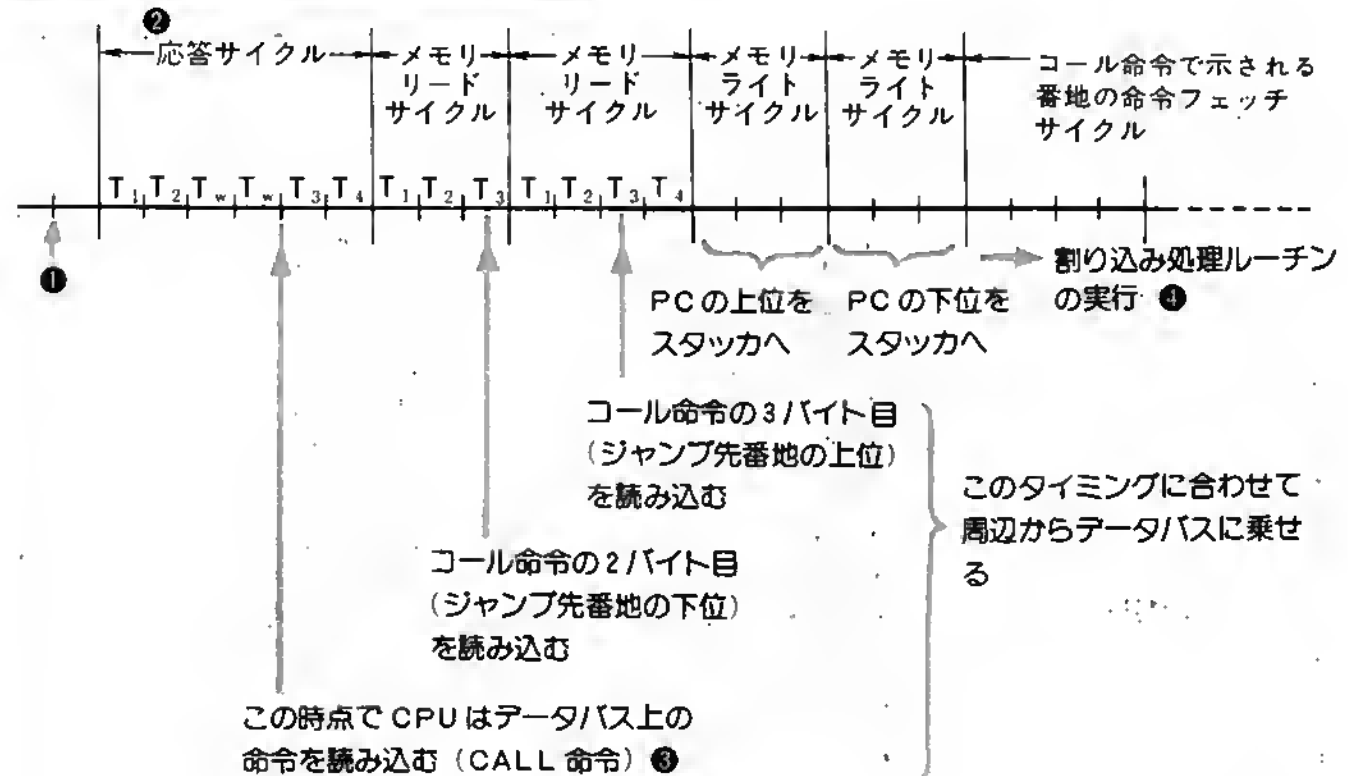


- ① 割り込み信号を与える
- ② 受け付けた返事（応答サイクル）
- ③ データバスを通じて命令コードを与える
- ④ CPUは③で受け取った命令を実行する

### リスタート [RST] 命令を与えた場合



### コール [CALL] 命令を与えた場合



## 37 モード1のインタラプト

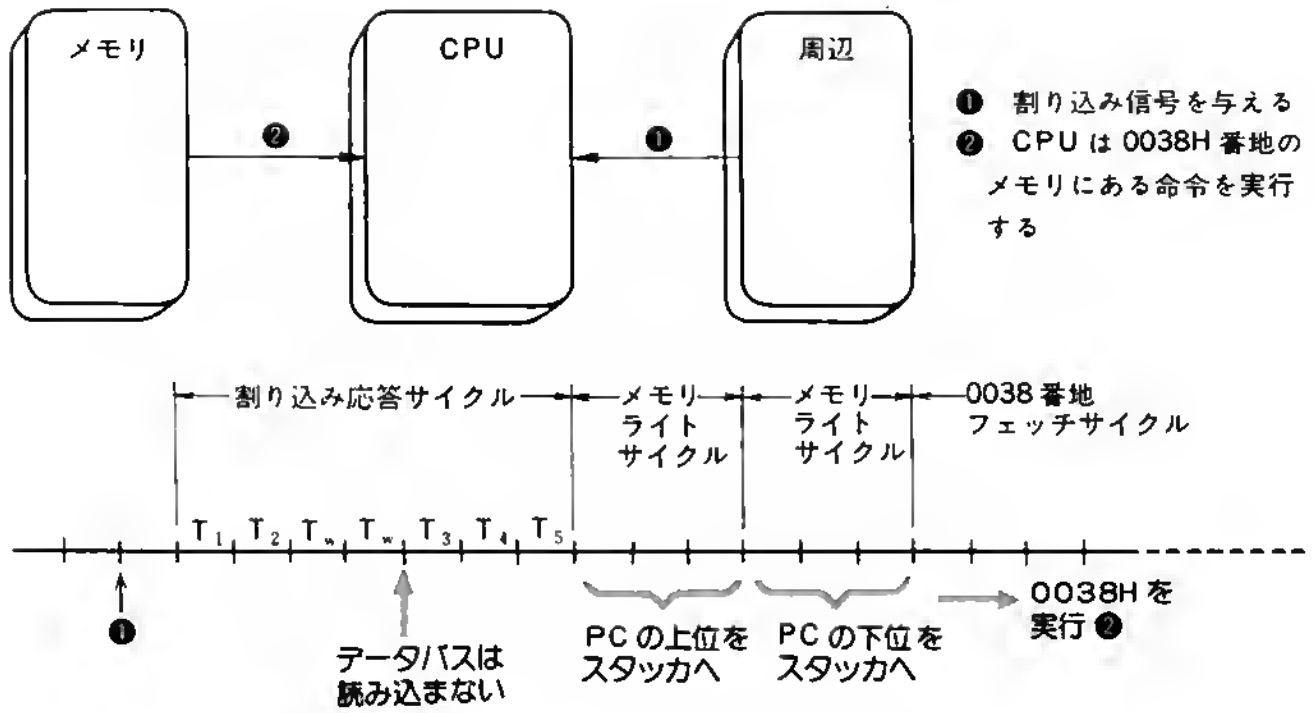
割り込みモード1は最も簡単な割り込みで、ほとんどノンマスカブル インタラプトと同じ動作をします。

インタラプト (INT) 信号の検知と、応答サイクルはモード0と同じです。応答サイクルでは、外部からは何も与えず、CPU も何も読み込まないので、ダミー サイクルとなります。次に現在のプログラム カウンタの値、すなわち戻り番地をスタッカへ格納するためにメモリ ライト サイクルが2回続き、そのあとプログラム カウンタを0038Hにします。次はこの番地の命令実行サイクルへ入ります。見かけ上は、[CALL 0038H]を実行したことになります(ただし、タイミングは異なります)。

このモードでは、割り込み要因によって、ジャンプ先の処理ルーチンを分けることはできません。もしこれが必要なときは、0038H 番地からのルーチンの中で、外部からのステータスを読み込み、解析しなければなりません。

Z-80 ファミリの呼び方

	ザ イ ロ グ		シ ャ ー プ	
2.5 MHz バージョン	Z-80	CPU	LH	0080
	Z-80	PIO	LH	0081
	Z-80	CTC	LH	0082
	Z-80	DMA	LH	0083
	Z-80	SIO - 0	LH	0084
	Z-80	SIO - 1	LH	0085
	Z-80	SIO - 2	LH	0086
4 MHz バージョン	Z-80 A	CPU	LH	0080 A
	Z-80 A	PIO	LH	0081 A
	Z-80 A	CTC	LH	0082 A
	Z-80 A	DMA	LH	0083 A
	Z-80 A	SIO - 0	LH	0084 A
	Z-80 A	SIO - 1	LH	0085 A
	Z-80 A	SIO - 2	LH	0086 A



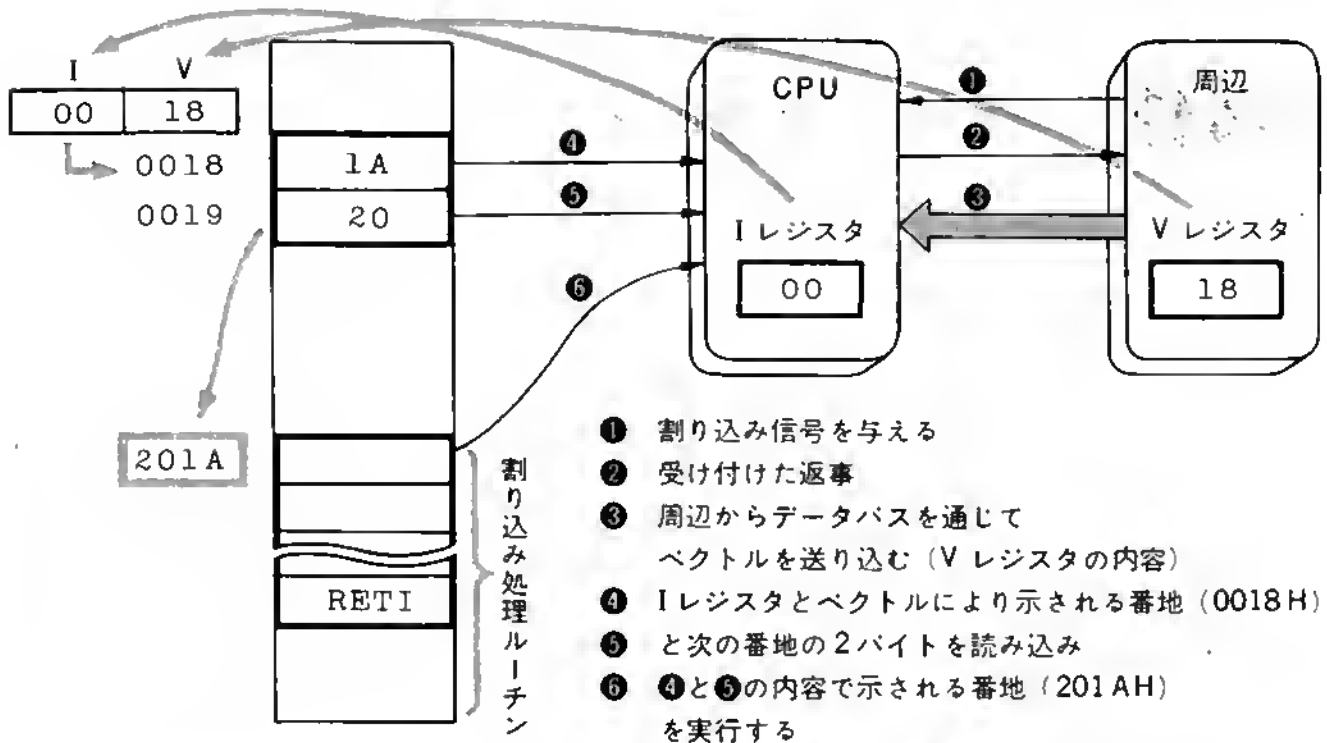
## 38 モード2のインタラプト

割り込みがかけられた次のフェッチサイクルは、割り込み応答サイクルとなります。他のモードと違うのは、エムワン ( $\overline{M1}$ ) 信号と IO リクエスト ( $\overline{IORQ}$ ) 信号が共に "L" になったら、ペリフェラルからはベクトルと呼ばれる 1 バイトの値をデータバスに乗せ、CPU がこれを受け取ることです。次に戻り番地として現在のプログラムカウンタの値をスタックへ格納するためのメモリライトサイクルが 2 回続きます。その次のサイクルでは、CPU は 2 バイトの情報をメモリから読み込むためのメモリリードサイクルが 2 回続きます。そしていま読み込んだ値をプログラムカウンタへ入れて、そのアドレスからの実行を開始します。

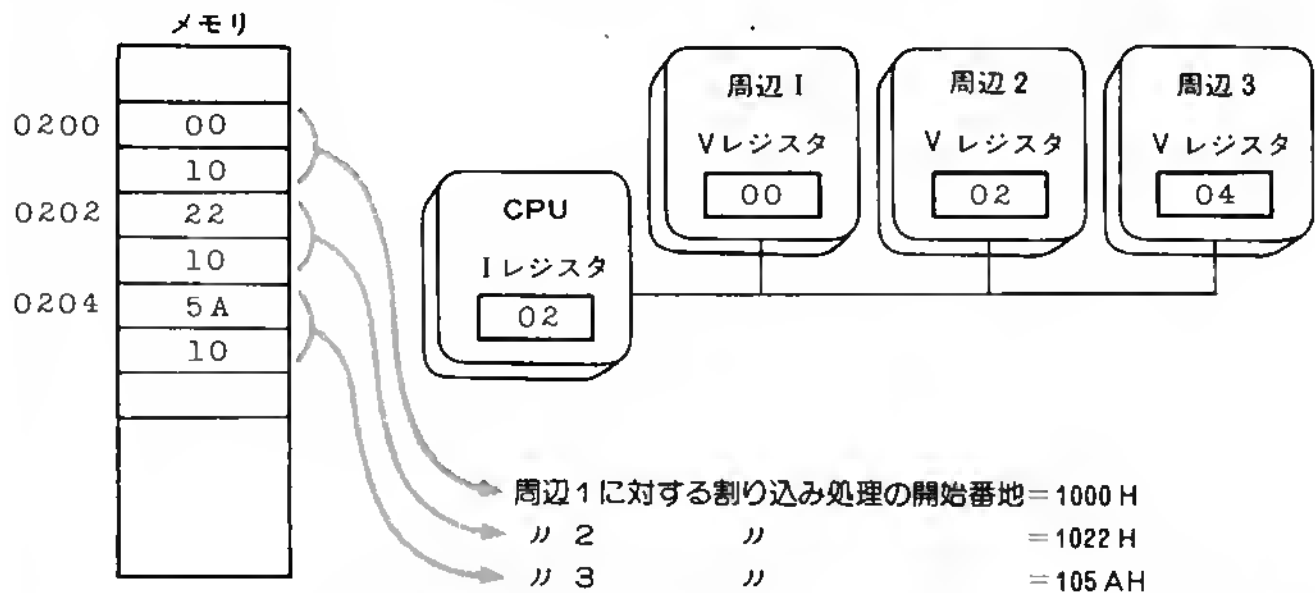
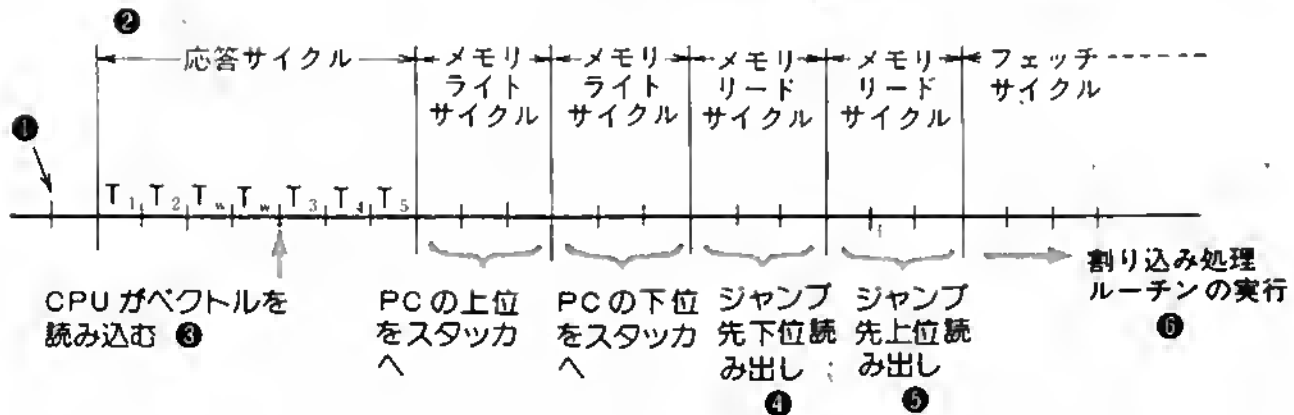
ベクトルとは、各ペリフェラルにあらかじめ書き込んである 1 バイトのデータで、これがアドレスの下位 8 ビットとなります。上位 8 ビットは、CPU 内の I レジスタへ、これもあらかじめ書き込んでおきます。割り込みがかかると、I レジスタの内容とベクトルで構成される番地と、次の番地のメモリの内容を 2 バイト読み込んで、この値の番地へジャンプすることになります。

ベクトルの値は、ペリフェラル個々に異なった値を書いておけば、割り込みをかけたペリフェラルによって別々の処理ルーチンへジャンプしますし、DMA や SIO では、割り込み発生の原因によってベクトルを変化させて送り出しますので、処理ルーチンの組み立てが簡単にできます。

このモード 2 の割り込み機能は Z-80 の大きな特徴です。ファミリと合わせてシステムを構成すれば、外部回路なしで優先順位の決定を含めた、割り込みシステムが完成してしまいます。



(上記の番地の数値は一例でありこの関係を保てば全メモリのどの番地でもよい)



このような表を割り込み処理ルーチン・エントリー・アドレス・テーブルと呼ぶ

割り込みの優先順位の決定にはデージーチェーン（ひな菊の輪）という手法がとられます。CPU に対して割り込み信号を出すペリフェラルを直列に継いでいき、割り込みをかけて処理ルーチン実行中のペリフェラルが、自分より下位のペリフェラルに対して割り込み禁止を順次伝達していきます。自分より上位の割り込みがかかると、自分の処理が保留されますので、これもデージーチェーンを通じて知ることができます。処理が終ってリターン フロム インタラプト [RETI] 命令を CPU がフェッチすると、データバスを横から監視している処理中のペリフェラルは自分の処理が終ったことを知り、下位のペリフェラルに対して割り込みの禁止を解除します。

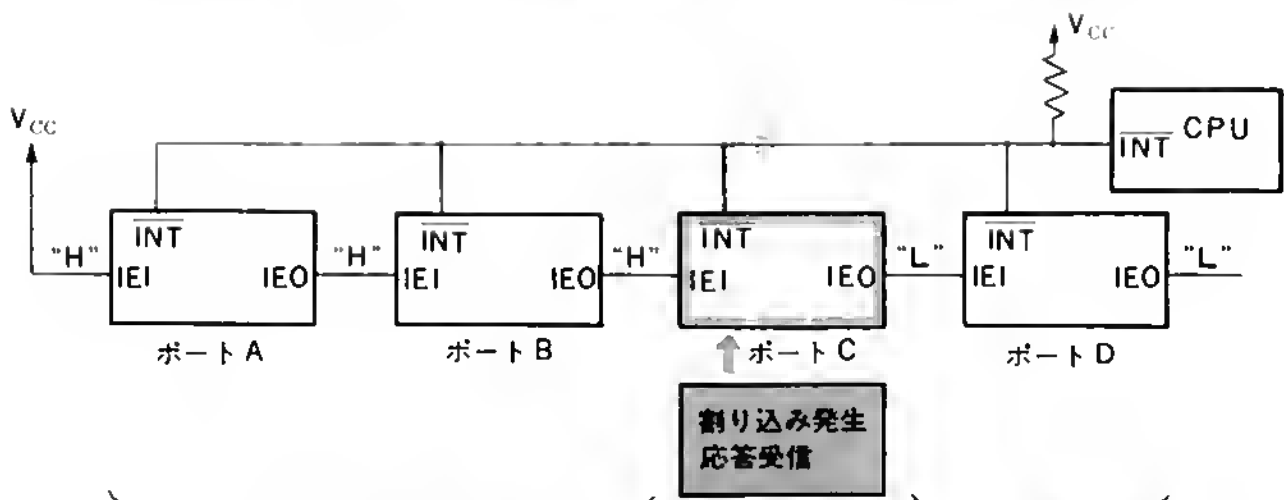
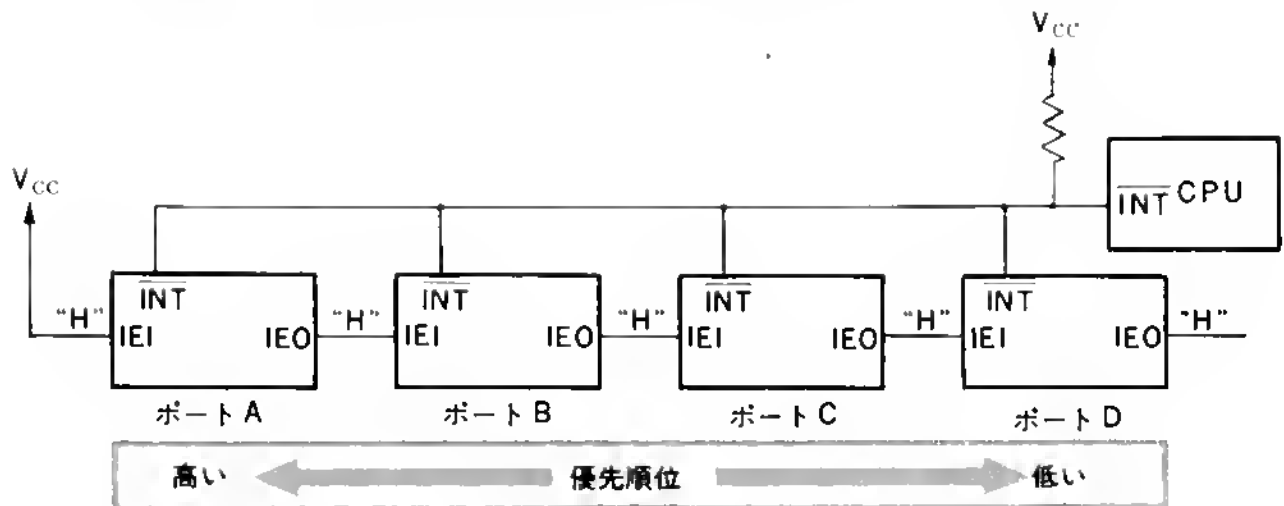
各ペリフェラルはインタラプト イネーブル イン (IEI) とインタラプト イネーブル アウト (IEO) 信号の端子を持っており、IEO を自分より下位の IEI に継ぎます。最も優先順位の高いペリフェラルの IEI は  $V_{CC} (+5V)$  へ継ぎ、最も低いペリフェラルの IEO はオープンにしておきます。

多数のペリフェラルを継ぐときは、伝達に時間がかかりますので、工夫がいります。そのまま継げるのは 4 個までです。

この方式は、割り込みだけでなく、マルチ CPU としたときや、DMA を複数使用するときのバスリクエストでも使われます。

### デー ジ ー チ ェ ー ン 端 子 ( 入 力 = I E I   出 力 = I E O ) の 機 能

- ① 通常 I E I が “H” になっていれば I E O も “H” が出ている。
- ② 割り込みを発生し ( $\overline{\text{INT}}$ ) 応答サイクルに入ると、割り込みを発生した周辺は I E O を “L” にする。
- ③ I E I に “L” が入ると I E O も “L” にする。
- ④ 割り込みは、I E I が “H” になっているときだけ発生し、“L” のときは発生しない。またすでに発生した後に I E I が “L” になったときは保留されたものとする。



このときも割り込み要因があれば CPU に対して割り込みを発生できる。  
 このときは優先順位の低いポートは割り込み中であっても保留させられ、高いほうを先に処理する

割り込み処理終了の命令 [RETI] を CPU がフェッチしたことをデータバスを見て知ると、I E O を “H” にもどす

これより優先順位の高いポートが割り込み中なのでおわるまで待たされる

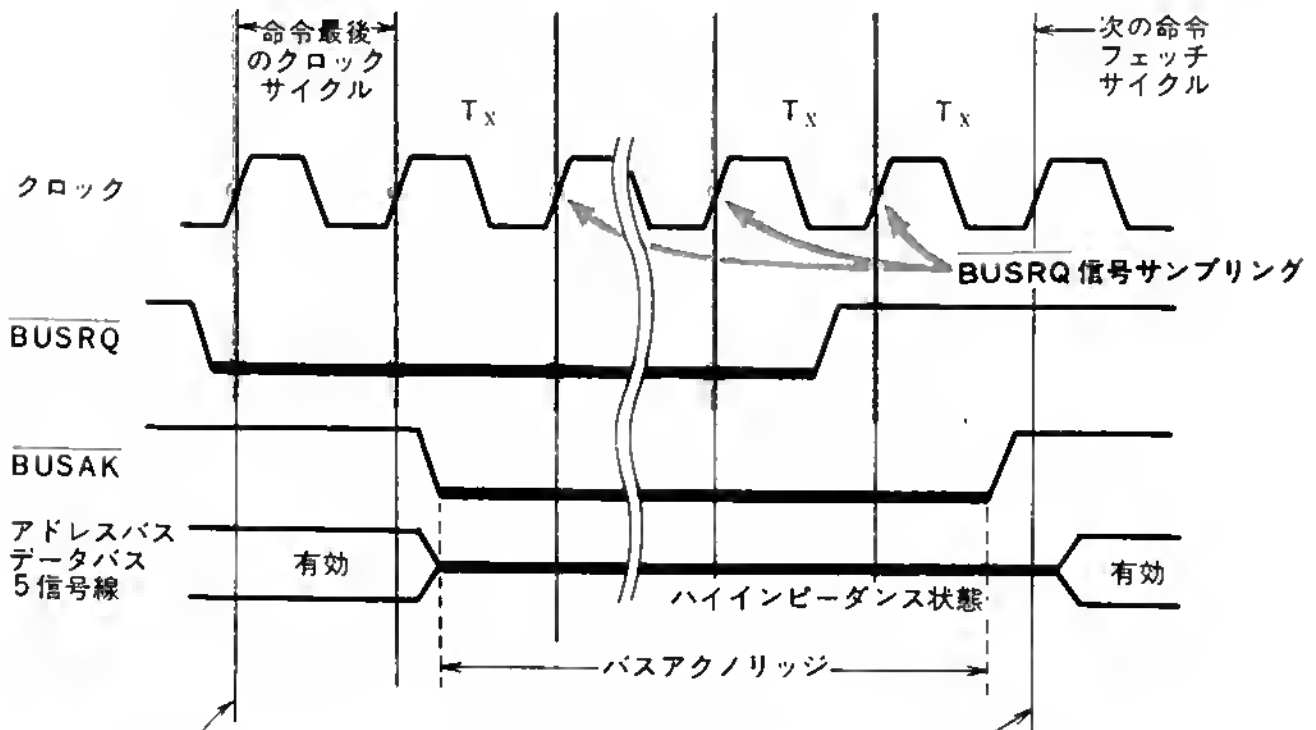
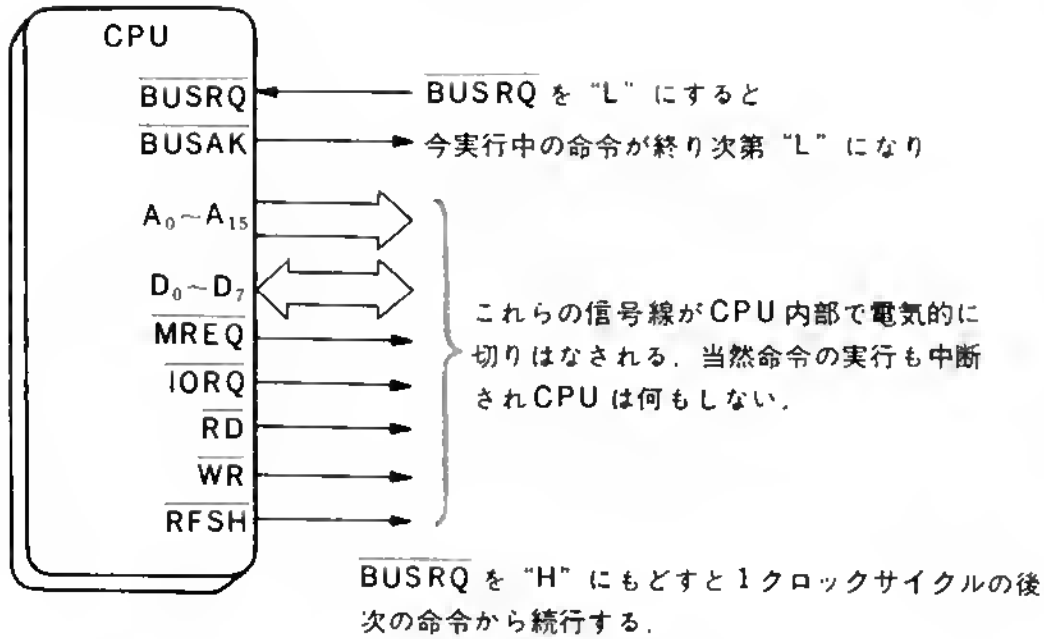
## 4.1 バス要求と応答

通常 CPU はバスを占有して動作していますが、マルチ CPU システムで他の CPU からの要求や、その他いろいろな理由で一時的動作を中止してバスを開けたいことがあります。このようなとき **バスリクエスト (BUSRQ)** 信号を与えると、トリスレータの端子をすべて高インピーダンスにし、**バスアクノリッジ (BUSAK)** 信号をアクティブにして外部に対して、バスを開け渡したことを知らせてきます。この間メモリリフレッシュは行なわれなくなりますので、長時間この状態を続けるときは注意が必要です。

バスリクエスト信号は、割り込み信号と同様に、命令の最後のクロックサイクルの立ち上りの時点で検知されます。ここでバスリクエストが“L”であると、次の命令のフェッチサイクルはなくなり、バスアクノリッジがアクティブになります。これ以後各クロックの立ち上りでもバスリクエストが監視され“H”に戻っていると、次のクロックサイクルから元の動作に戻ります。バスアクノリッジがアクティブな期間は、ノンマスカブルインタラプトとインタラプトは受け付けられなくなります。しかしバスリクエストが最初に検知される時点での両割り込みは受け付けられ、バスアクノリッジが解除後に、それぞれの動作に入ります。

バスリクエスト端子を使用しないときは、 $V_{cc}$  へ継いで“H”レベルにしておいて下さい。





ここまで入っていた  $\overline{\text{NMI}}$  はバスリクエスト解除後に実行する。  
 バスアクノリッジ中は  $\overline{\text{NMI}}$ ,  $\overline{\text{INT}}$  は無視される。  
 リフレッシュもされない。

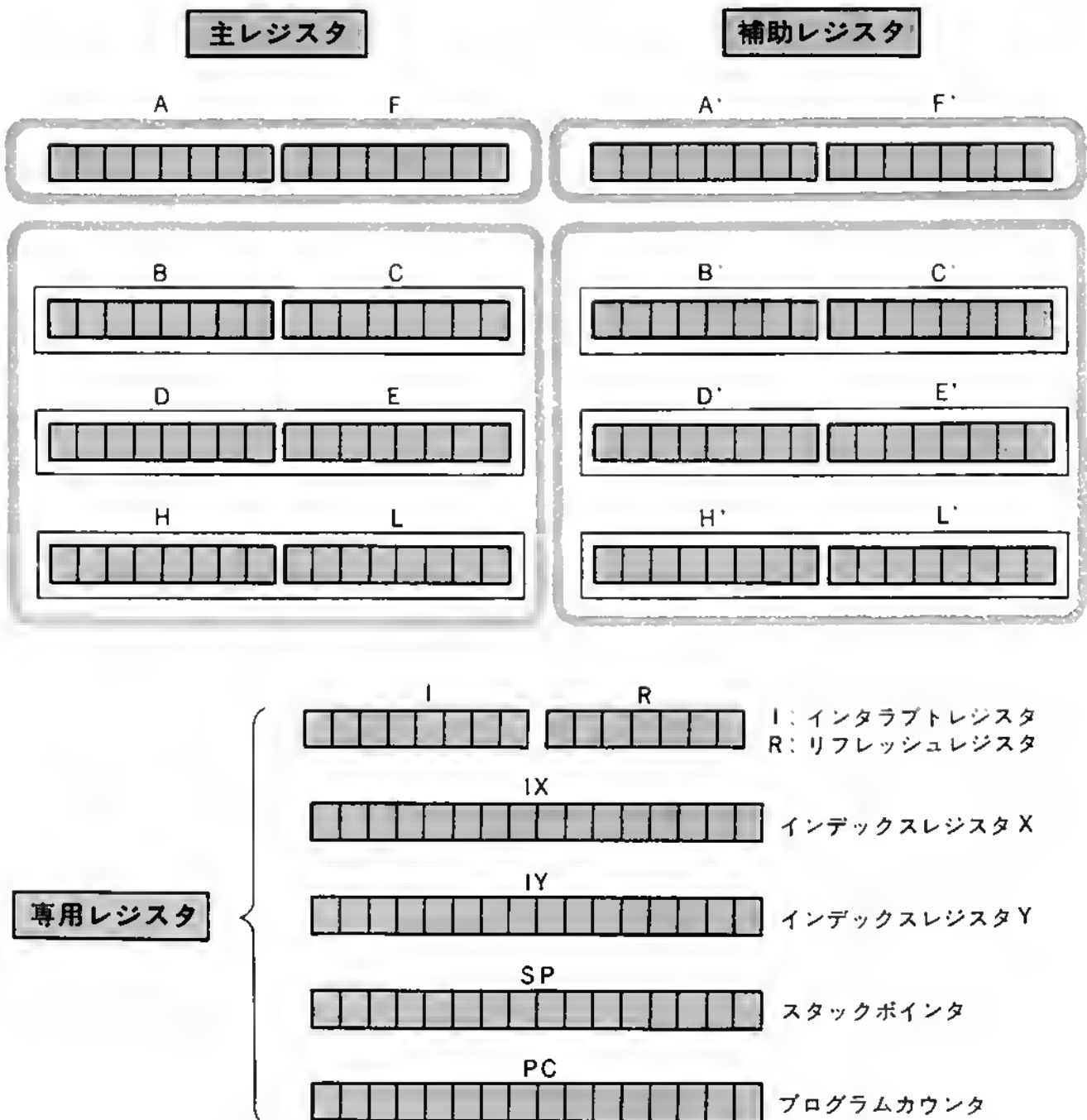
## 4-1 内部レジスタの構成

CPU 内部には、種々の作業に使うための一時的なメモリがたくさんあります。1ビット単独のものは、フリップフロップ (F/F) と呼ばれ、複数ビットが一連にして使われるものは、レジスタと呼ばれます。このうちでユーザに公開されるレジスタが22本あります。それぞれに個性を持っていて、うまく使いわけることが、よいプログラムを作る上で重要です。どのレジスタをどう使うかは一概にはいえず、ケースバイケースで考えなければなりません。プログラムをいくつも作るうちに、だんだんとうまくなっていくでしょう。

8ビット単独で働くレジスタは、AレジスタとFレジスタです。Aレジスタはアキュムレータとも呼ばれ、最も重要な働きをします。Fレジスタはフラグレジスタとして各ビットに意味を持つレジスタです。8ビット単独でも、二つを継いで16ビットとしても働くレジスタは、B、C、D、E、H、L、です。ここまでの8本のレジスタは、主と補助の2組あり、内容を変換することができます。

専用レジスタとして用途が決められているものは6本あります。割り込み処理で使われるIレジスタは8ビット、メモリリフレッシュアドレスをカウントしているRレジスタは7ビットです。作表に便利なインデックスレジスタはIX、IYの2本あり、各16ビットです。スタッカのアドレスを入れておくスタックポインタとプログラム実行アドレスをカウントするプログラムカウンタも16ビットです。

16ビットレジスタには2桁、8ビットレジスタには1桁の名称が付けられており、補助レジスタには' (ダッシュ) を付けて、たとえばA'と呼ぶようになっています。



## 42 A, I, R, F レジスタ

レジスタ群の中で最も多用されるAレジスタは、人が手作業で計算するときのソロバンにたとえられます。加算を行なう場合を例にとると、まず足される数をAレジスタへ入れます。次にAレジスタに対して、別の足す数を加えると、Aレジスタの内容が答になるわけです。このような用途のレジスタを**アキュムレータ**と呼びます。

8ビットの加算、減算、論理和、論理積、排他的論理和、比較は、すべてAレジスタの内容に対して行なわれます。

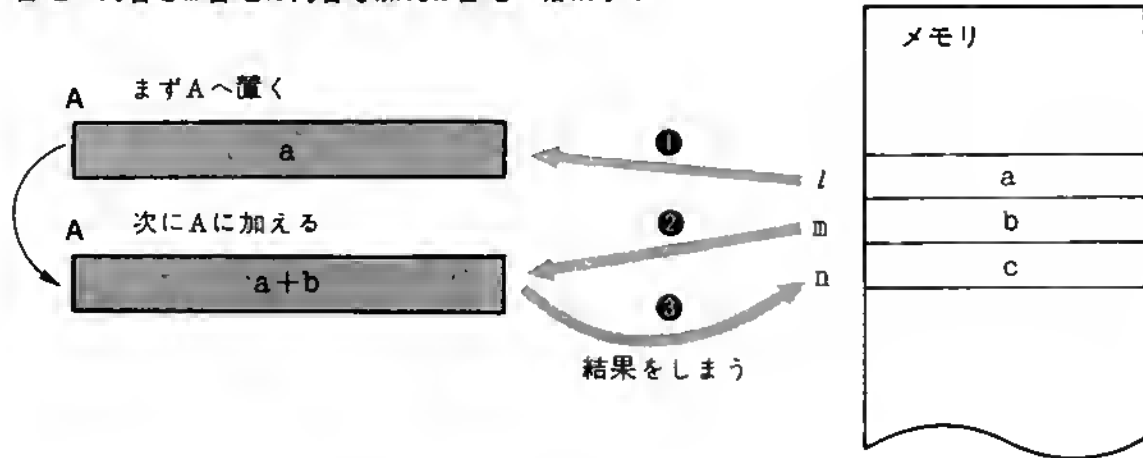
IレジスタとRレジスタに読み書きするときは、一度Aレジスタを経由しないとできません。入力、出力も、ポートアドレスに絶対番地を指定するときは、Aレジスタを経由します。

Iレジスタは、割り込みモードを2に指定したときに、割り込み処理ルーチンへジャンプするためのジャンプ先のアドレス表のあるアドレスの上位8ビットを表わします。初期値設定のために1回書き込むだけで、ほとんどいじる必要はないレジスタです。

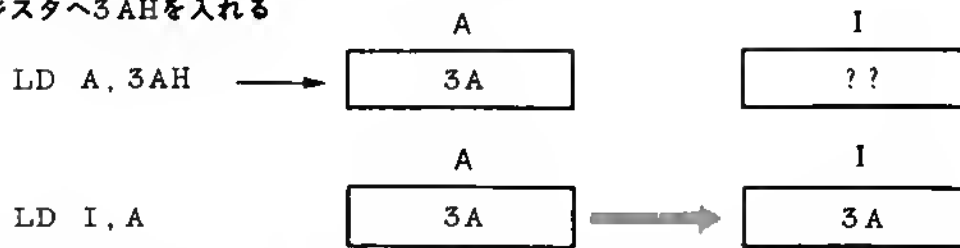
Rレジスタは、リセット信号でゼロになり、フェッチサイクルごとに1ずつ増加して、リフレッシュ用のアドレスを指定しています。プログラムで内容を変えることはほとんどありません。むしろ正確にリフレッシュさせるためには、さわらないほうがよいのです。

Fレジスタは、プログラムで内容を変えることはできません。1命令実行ごとに、その命令により出てきた結果に特別の意味があれば、それ以後の命令実行に必要な情報を記憶します。たとえば、減算命令を実行して結果がゼロであれば、Fレジスタ中の7番目のビット（これをゼロフラグといいます）を“1”にしてこのことを憶えておきます。次の命令が、「結果がゼロならばジャンプせよ」といった命令である場合、これをチェックして判断するようになっています。このように種々のフラグ（旗）が並んでいるレジスタがFレジスタです。

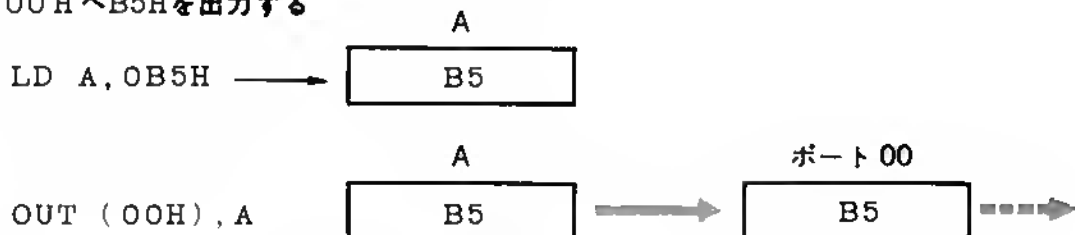
- l 番地の内容と m 番地の内容を加え n 番地へ格納する



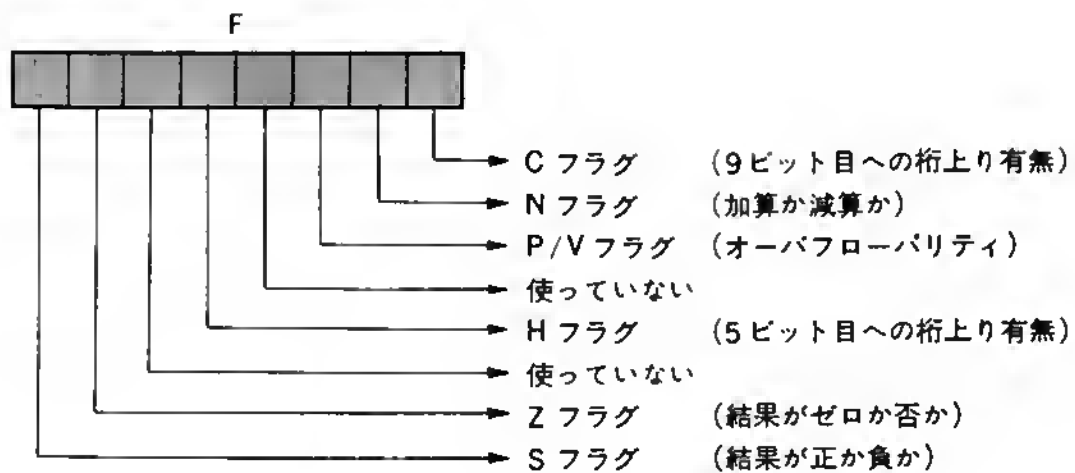
- I レジスタへ 3AH を入れる



- ポート 00H へ B5H を出力する



- F レジスタ



## 4.3 汎用レジスタ

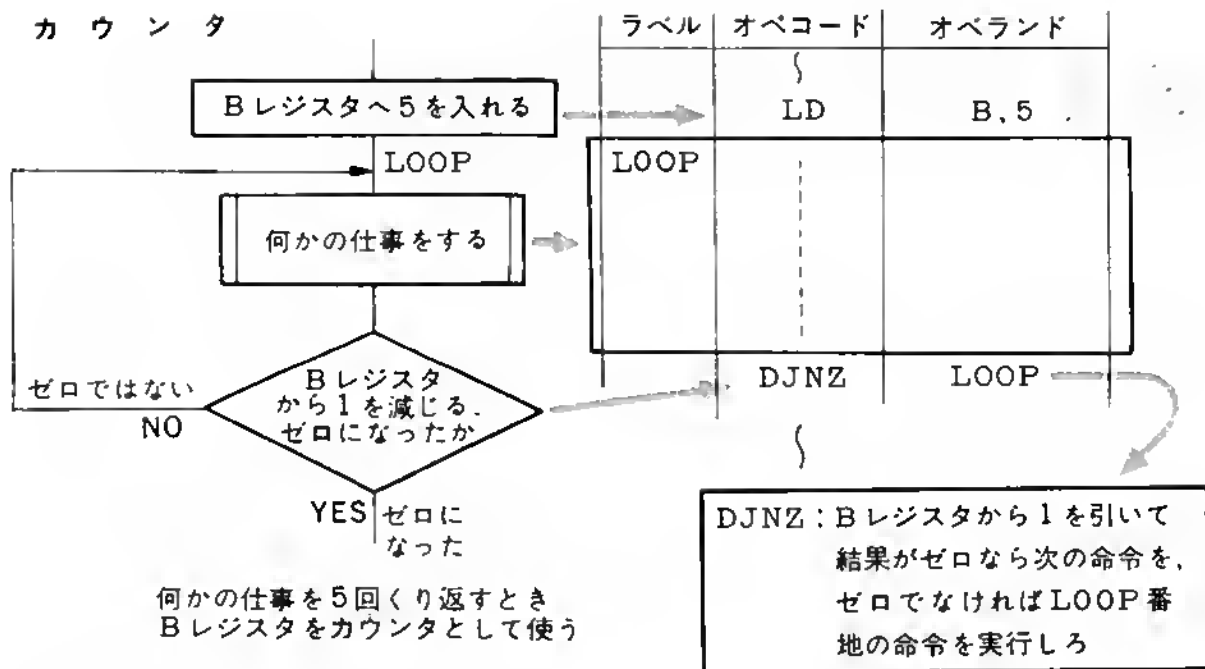
汎用レジスタには、H、L、B、C、D、Eの6本と、同数の補助レジスタがあります。2本のレジスタをペアにして、HL、BC、DEと表現し、16ビット レジスタとして使うこともできます。

8ビット レジスタとしての働きは、主に演算途中の数値などを一時記憶することですが、Bレジスタはループ回数を入れておき、1回に1ずつ減算し、ゼロになったら、ループから出るというカウンタの役目をさせることがあります。このためにデクリメント ジャンプ ノンゼロ [DJNZ] 命令がとても便利です。Cレジスタは入力[IN]、出力[OUT]命令で、ポート アドレスを指定するポインタとして使えます。

16ビット レジスタとしてペアにして使うと、HLは加算減算でアキュムレータとして働きます。また、アドレスを入れ、間接的にアドレス指定をすることができます。

繰り返し処理でアドレスを入れておくレジスタを**ポインタ**、回数を入れておくレジスタを**カウンタ**と呼びます。ブロック転送 [LDIR] 命令は、元のアドレスをHL、先のアドレスをDE、ブロックの長さ(バイト数)をBCに入れてから実行すると、HLの内容番地のメモリからDEの内容番地のメモリへ1バイト転送し、HLとDEに1を加え、BCから1を引きます。これを繰り返しBCがゼロになると実行を終る、という命令です。HLとDEをポインタとして、BCをカウンタとして使用しています。

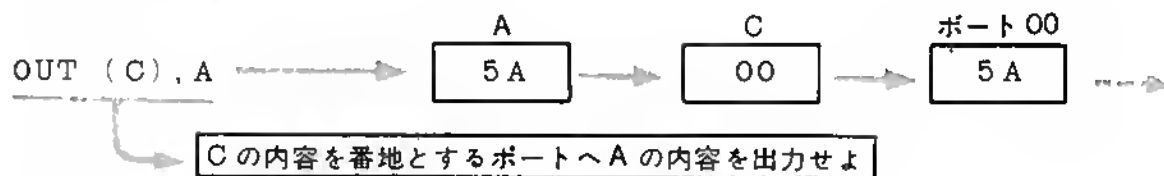
## ● カ ウ ン タ



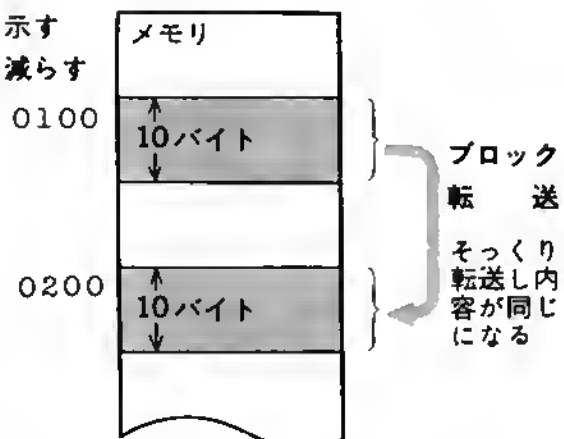
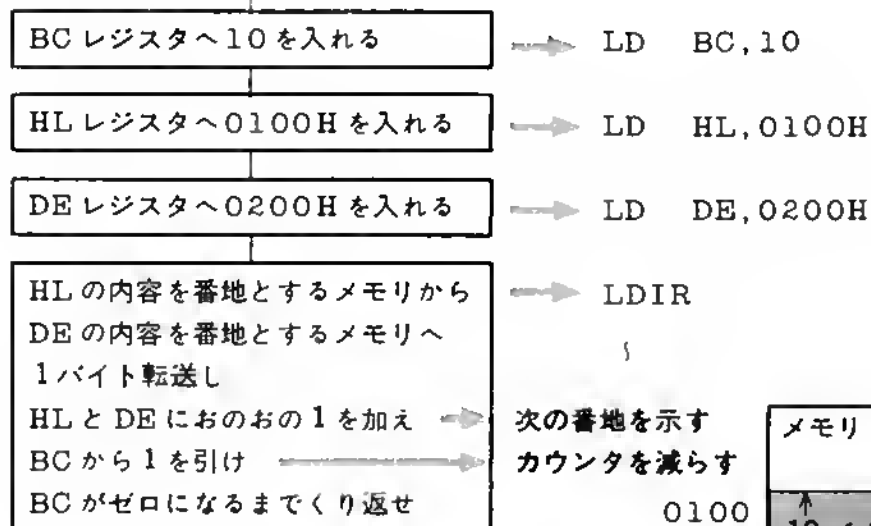
## ● ポ イ ン タ

Cレジスタの内容が 00H

Aレジスタの内容が 5AH とすると



## ● ポインタとカウンタ



## 4.4 補助レジスタと交換命令

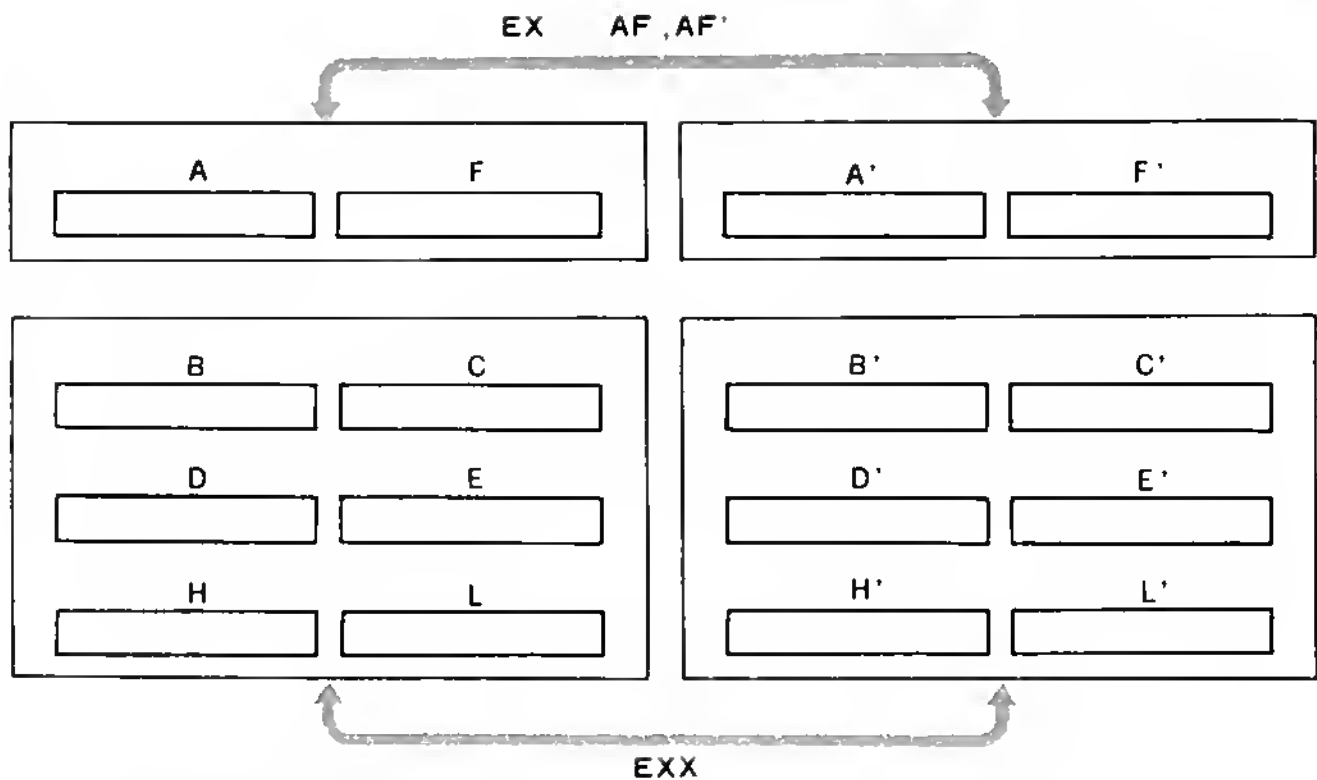
A, F, H, L, B, C, D, E の各レジスタは、同じものがもう 1 組あります、**補助レジスタ**と呼び、主レジスタとは区別して、“'” ダッシュをつけて表現します、補助レジスタの内容は、交換命令と呼ばれる一連の命令で、主レジスタの内容と入れ替える以外に操作することはできません、したがって、主レジスタの裏側にあって、交換命令でひっくり返すと、表（主）へ出てくると考えられます、補助レジスタとの交換命令には、AF と AF' のペアを交換する[EX AF, AF'] と、上記以外の汎用レジスタを交換する [EXX] の二つがあります、

サブ ルーチンや割り込み処理ルーチンの中でメイン ルーチンで使用中のレジスタの内容を変えたくないことがあります、このようなときは、サブ ルーチンに入ったところで、レジスタの内容をメモリへ待避させてから、必要なレジスタを使い、サブ ルーチンから出る前に待避した内容をレジスタへ戻せばよいのですが、交換命令を使えば、1 命令で全レジスタを待避させることができます、特に、スピードが問題になる割り込み処理の場合は都合のよい命令です、ただし、メイン ルーチンで主レジスタ、サブ ルーチンで補助レジスタを使うことをきちんと取り決めておかないと、どちらに何が入っているのかわからなくなります、

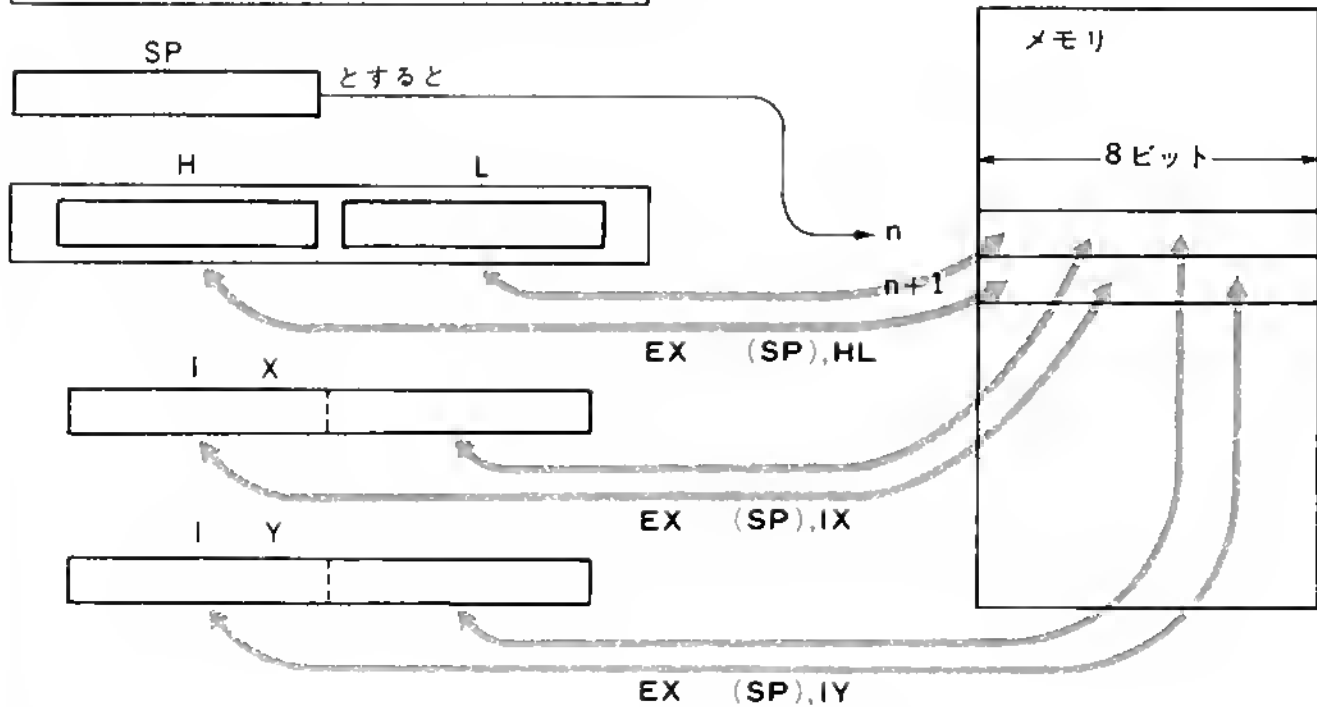
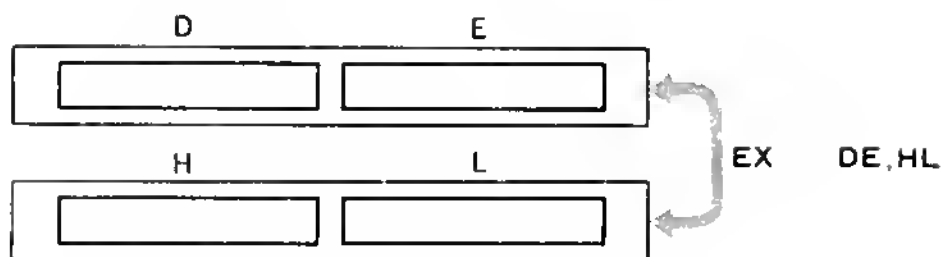
交換命令には他に DE と HL を交換する [EX DE, HL] と、HL, IX, IY の各内容とスタッカの内容を交換する[EX (SP), HL], [EX (SP), IX], [EX (SP), IY] があります、

なお、通常の転送命令の場合は、たとえば [LD A, B] であれば、B レジスタの内容が A レジスタへ転送されるだけで、元の A レジスタの内容はなくなってしまいます、B レジスタは元のまま変わりません、交換命令では双方ともこわれることなく入れ替わります、





EX, EXX 以外の命令では  
主レジスタしか扱えない



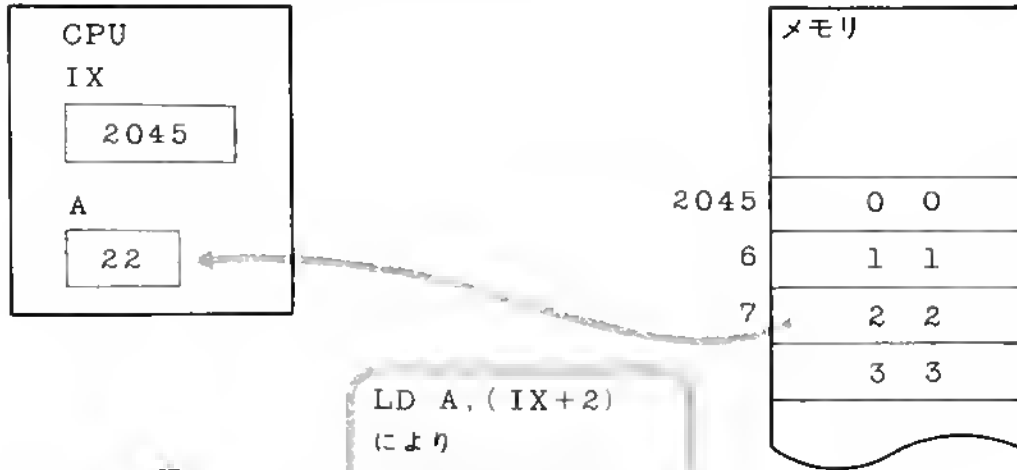
## 45 IX, IYレジスタ

IX, IYレジスタは、**インデックス レジスタ**と呼ばれ、同一の働きをする16ビットレジスタが2本あります。通常はここへメモリのアドレスを入れて、間接的にアドレス指定をするときに使います。表を操作するときなどに表の先頭アドレスを入れておき、1ずつ増加させながら次々に読み出すことができます。また、先頭から何番目のデータが欲しいというようなときは、先頭からの距離をプログラムで指定することもできるレジスタです。たとえば、`[LD A, (IX+5)]`と書けば、IXに入っている数プラス5番地のメモリ内容をAレジスタへ入れることができます。距離は、-128～+127バイトまでの範囲に限られます。距離のことを**ディスプレイスメント**と呼びます。

このレジスタの内容を間接アドレスとしてソースまたはディスティネーションに指定した命令のマシン語は、オペコードが2バイト構成になります。3バイト目には、ディスプレイスメントの値が入ります。オペランドがある場合は4バイト目に続きます。

実行のサイクルは、ディスプレイスメントの演算のためのクロック・サイクルが入るため、例外的に長くなります。`[LD A, (IX+5)]`では、19クロックサイクルです。

たとえば IX レジスタに 2045H が入っていると



LD A, (IX+2)のマシン語  
DD オペコード  
7E オペコード  
02 ディスプレイスメント

LD A, (IX+2)  
により  
A レジスタには IX+2  
すなわち 2047H 番地  
の内容が転送される

メモリに 3 バイトを一組とするデータ  
があるとする

メモリ	
2040	①
1	②
2	③
2043	①
4	②
5	③
2046	

```

LD  IX, 2040H      最初の組の先頭番地を IX へ
LD  DE, 3          DE へ 3 を入れておく
LOOP LD  A, (IX)    1 バイト目を A レジスタへ
      LD  B, (IX+1)  2 バイト目を B レジスタへ
      LD  C, (IX+2)  3 バイト目を C レジスタへ

      A, B, C レジスタにある
      1 組を処理する

      ADD IX, DE     次の組の先頭番地を IX へ
      JP  LOOP
  
```

## 4.6 スタックとスタックポインタ (SPレジスタ)

サブルーチンコール [CALL] 命令を実行したり、割り込みルーチンへジャンプしたりするときに、戻り番地を自動的に記憶する機能があることは前述しました。これ以外にもプログラム中で一時的にレジスタの内容をメモリに退避したいことがよくあります。このようなときに大変便利なのがスタックです。

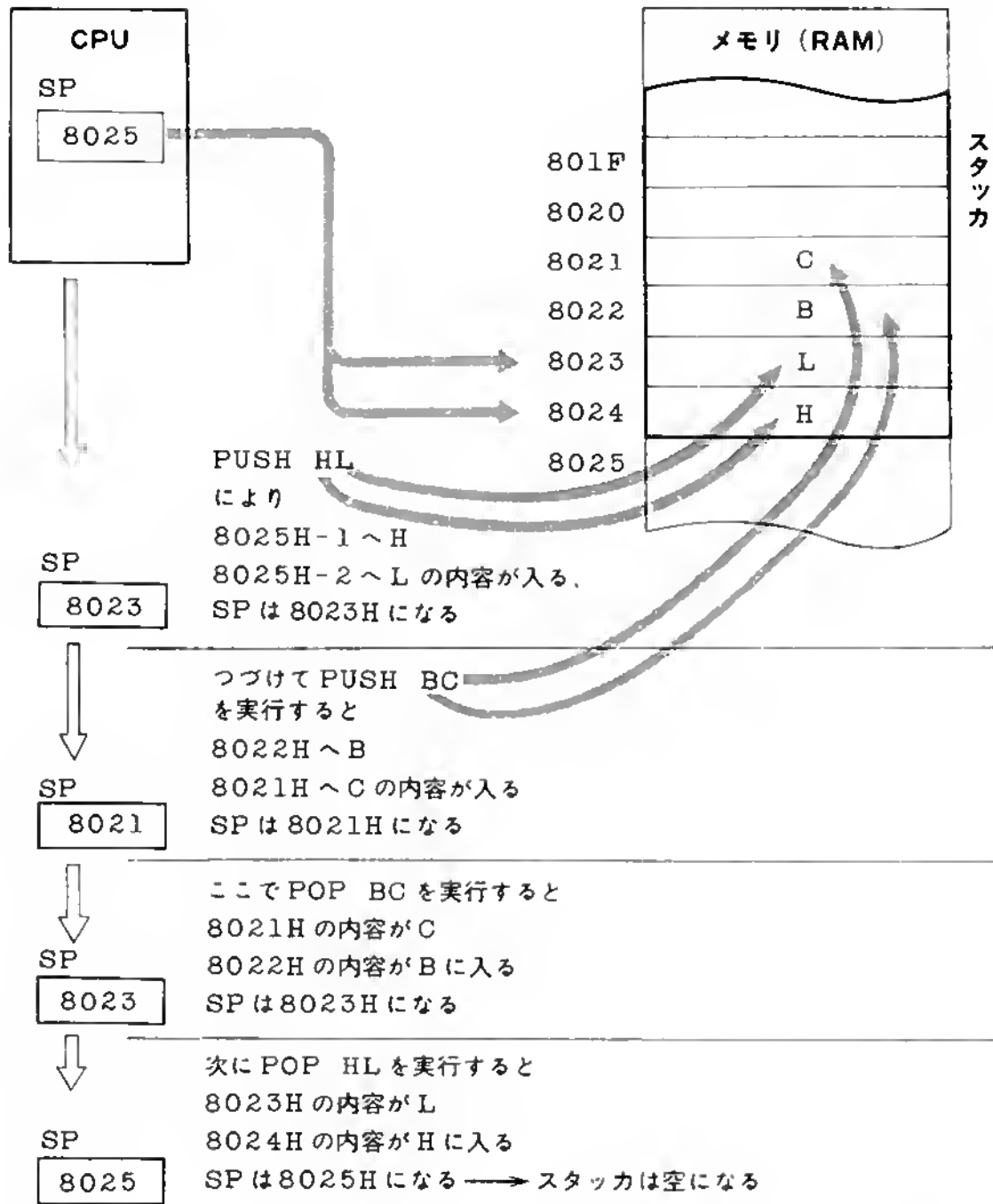
**スタック**はメモリ (RAM) の一部を割りあてた記憶領域のことですが、このアドレスは、**スタックポインタ (SP)** と呼ばれる 16 ビットレジスタに入れておきます。

スタックポインタに 8025H という値がプログラムによって入れられているとしましょう。このとき [PUSH HL] を実行すると、8025H-1 の 8024H へ H の内容が、8025H-2 の 8023H へ L の内容が書き込まれスタックポインタは 8023H になっています。次に読み出すときは [POP HL] を実行すると、8023H の内容を L に、8023H+1 の内容を H に転送し、スタックポインタは 8025H になります。

スタックに次々にしまい込まれた値は後から入れたほうが先に読み出されますので、**First In Last Out (FILO)** タイプのスタックと呼ばれます。サブルーチンや割り込みルーチンに入るときは自動的に [PUSH PC]\* に相当する命令が実行され、戻るときは [POP PC]\* に相当する命令が実行されていることになります。したがってサブルーチンの中から次のサブルーチンをコールするときも、(これをサブルーチンの**ネスティング**といいます) うまく働いてくれます。ただし、スタックとして割りあてた領域を超えるネスティングはできません。未完成のプログラムであやまって次々にサブルーチンコールをすると、このようなことが起こることがあります。プログラム設計の際に、スタックの大きさを決定するときは、慎重な判断を要します。またプログラムの始まりでは必ずスタックポインタのイニシャライズ (初期値設定) を行なって下さい。[PUSH], [POP] できるペアレジスタは AF, BC, DE, HL, IX, IY です。

---

\* この命令は、プログラム上では使えない。



プログラム中ではいちいち SP の値を気にしなくてもよい。  
ただし、PUSH と POP は対にして使わなければならない  
(意図した変則的な場合もある)

この間で H, L, B, C の内容は変わっても  
AA での内容と BB での内容は同じになっている。

CALL では PUSH PC  
RET では POP PC が実行されているのと同じなので、  
かならず対にして使う

AA  
PUSH HL  
PUSH BC  
POP BC  
POP HL  
BB

## 4.7 転送命令

最も基本的な転送命令は、ロード命令 [LD a, b] です。オペランドにはソースとディスティネーションの指定をします。ニモニックは Z-80 ではすべて [LD] で統一されています。8 ビットの転送では、ソースにはレジスタ名、ペアレジスタによる間接アドレス、インデックス レジスタによるディスプレイメント、直接アドレス値および定数が指定できます。ディスティネーションには、ソースで使用できるうちの定数以外を指定できます。ただし、I, R レジスタは A レジスタとのやりとりしかできないなど、ソースとディスティネーションの組み合わせには限定があります。

16 ビット転送では、ペア レジスタとやりとりのできる組み合わせは、16 ビット定数、直接アドレス値だけで、ペア レジスタ間のやりとりは、HL, IX, IY からスタック ポインタ (SP) への転送だけになります。

アセンブリ語で記述する場合、アドレス値や定数は、10 進数、16 進数で書いてもよいのですが、ラベル名で記述することもできます。マシン語になおしたときは、アセンブルリスト上は、16 進数に統一されます。16 ビット定数やアドレス値のときは、オペコードの次に下位 8 ビット、その次に上位 8 ビットがくるように並びます。

レジスタ名や定数に ( ) カッコを付けたオペランドは、メモリへまたはメモリからの転送命令を意味します。カッコ内の数値やラベルは、メモリのアドレスを指しています。16 ビット転送の場合は、このアドレスが下位 8 ビット、次が上位 8 ビットの転送アドレスになり、ペア レジスタとの転送では、たとえば HL ならば、H レジスタが上位、L レジスタが下位に対応します。

[PUSH], [POP] も 16 ビット転送命令ですが、スタック ポインタの項で述べました。

## 8ビットロード

ディスティネーション (受け側)		ソ ー ス (送 り 側)		
		レ ジ ス タ	メ モ リ	定 数
レジスタ	A	A, B, C, D, E, H, L, I, R	(HL), (BC), (DE), (IX+d), (IY+d), (定数)	定数
	B	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	定数
	C			
	D			
	E			
	H			
	L			
	I R	A	—	—
メモリ	(HL)	A, B, C, D, E, H, L	—	定数
	(BC) (DE)	A	—	—
	(IX+d) (IY+d)	A, B, C, D, E, H, L	—	定数
	(定数)	A	—	—

アドレス

Aレジスタはすべての  
転送ができる

メモリ→メモリの転送はできない

8ビット定数

## 16ビットロード

ディスティネーション(受け側)		ソ ー ス (送 り 側)		
		レ ジ ス タ	メモリ	定 数
レジスタ	BC DE HL	—	(定数)	定 数
	SP	HL, IX, IY		
	IX IY	—		
メモリ	(定数)	BC, DE, HL, SP, IX, IY	—	—

PUSH, POP のできるレジスタ  
AF, BC, DE, HL, IX, IY

LD (nn), HLや

LD HL, (nn)

の場合

nn    へL(下位)    nn

nn+1へH(上位) nn+1

が対応する

メモリ

L

H

## 4.8 算術演算命令

レジスタやメモリ内の8ビットないし16ビットのビットパターンを2進数と見なして、数学的な演算をする命令があります。8ビットの演算は、アド[ADD]、サブトラクト[SUB]、アド ウィズキャリ[ADC]、サブトラクト ウィズ キャリ[SBC]の4命令があります。Aレジスタに対して、他のレジスタかメモリの内容を加減算します。この演算のとき桁あふれまたは桁がりをする命令と無視する命令に別れます。あふれた桁はFレジスタの1ビット目のキャリフラグ(Cフラグ)があてられます。

コンペア[CP]命令は、Aレジスタからオペランドの内容を減算しますが、Aレジスタの内容は変わらず、Fレジスタが減算のときと同じに変化し、あとの命令で、結果がゼロであったかチェックできます。ゼロであったときは、Aレジスタの内容とオペランドの内容が同じであると判断できます。

インクリメント[INC]命令は、オペランドに1を加える命令です。デクリメント[DEC]は、オペランドから1を引く命令です。メモリ内のデータ列を1バイトずつ順次操作する場合や繰り返し行なう仕事の回数をカウントするときに使います。

ニゲイト[NEG]は、Aレジスタの内容をゼロから引いてAレジスタに入れる命令です。すなわち、Aレジスタの内容の符号(プラス、マイナス)を反転させるとき主に使います。

16ビットの演算は、HL、IX、IYレジスタの内容に対して行なう加減算で、桁あふれを考えに入れない演算は加算だけです。インクリメント[INC]とデクリメント[DEC]は、8ビットと同様です。メモリアドレスの操作に使います。

算術演算命令でのFレジスタの働きは重要で、多桁の演算では、キャリフラグの働きに注意する必要があります。また10進補正[DAA]命令は、Fレジスタの結果を参照して補正のし方を決定します。



	命令	ディスティネーション 被演算数→答	ソ                      ス			操   作 s: ソース Cy: キャリ
			レジスタ	メモリ	定数	
8 ビット	ADD	A	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	定数	$A+s \rightarrow A$
	ADC	A				$A+s+Cy \rightarrow A$
	SUB	A*				$A-s \rightarrow A$
	SBC	A				$A-s-Cy \rightarrow A$
	CP	A*				$A-s$ Aは不変
	INC	ソースと同じ*	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	—	$s+1 \rightarrow s$
	DEC	ソースと同じ*				$s-1 \rightarrow s$
	DAA	A*	A*	—	—	Aを補正
	NEG	A*				$0-A \rightarrow A$
16 ビット	ADD	HL	BC, DE HL, SP	—	—	$HL+s \rightarrow HL$
	ADD	IX	BC, DE SP, IX			$IX+s \rightarrow IX$
	ADD	IY	BC, DE SP, IY			$IY+s \rightarrow IY$
	ADC	HL	BC, DE			$HL+s+Cy \rightarrow HL$
	SBC	HL	HL, SP			$HL-s-Cy \rightarrow HL$
	INC	ソースと同じ*	BC, DE, HL			$s+1 \rightarrow s$
	DEC	ソースと同じ*	SP, IX, IY			$s-1 \rightarrow s$

\*印はアセンブリ語ではディスティネーションはオペランドに書かない。(例) SUB C    INC A  
DAA NEG ではソースも書かない。

ADD    ADC

```

  10111001
+)01010101
-----
 100001110

```

桁上り Cフラグ(キャリ)へ入る

ADCではCフラグ(キャリ)もここへ加える。  
すなわち、さらに下の桁の演算がこの前に行な  
われているとすると、そのときの桁上りが含ま  
れる

SBCは、けたがりCフラグに入っているのをこれを含めての演算をする  
SUBは、前の演算でけたがりなかったものとしてCフラグは無視してしまう

## 49 論理演算命令

論理演算命令は、すべて8ビット単位に、Aレジスタに対して行なわれます。各ビットごとの論理積、論理和、排他的論理和、否定をとります。

アンド [AND] 命令は論理積です。Aレジスタとオペランドの内容の双方共に "1" のビットだけ "1" が残り、他は "0" になります。Aレジスタのある特定のビットだけ残したいときは、ここを "1"、他を "0" としたビットパターンをオペランドで指定して、アンドをとれば、不要なビットは必ず "0" になり、必要なビットだけが "1" であれば "1"、"0" であれば "0" として残ります。この方法をマスクといいます。Z-80ではビット操作命令があるので、あまり使いません。

オア [OR] 命令は論理和です。Aレジスタとオペランドの内容のいずれかが "1" のビットを "1" として、双方共に "0" のビットを "0" にします。16ビット演算のデクリメント [DEC] 命令でHLレジスタの内容を1ずつ減らしてゼロになってもフラグは変化しません。そこで、このチェックを行なうときは、HレジスタをAレジスタに移して、Lレジスタとのオア [OR] をとりますと、H、L、共に全ビットが "0" すなわちHLレジスタとしてゼロになっているときだけ結果がゼロになり、フラグレジスタのゼロフラグがこのことを示します。また、特定のビットを前の内容にかかわらず "1" にしたいときには、このビットが "1"、他が "0" のビットパターンとオア [OR] をとれば、"1" のところは "1" になり、"0" のところは前のまま残ります。

エクスクルーシブオア [XOR] は排他的論理和です。Aレジスタとオペランドの内容が、ビット単位に同じところを "0"、異なるところを "1" にします。AレジスタとAレジスタのエクスクルーシブオアをとるとAレジスタは必ずゼロになります。なおキャリフラグだけを "0" にするときは、[AND A]か[OR A]を使います。

コンプリメント [CPL] 命令はビットパターンをすべて反転させます。すなわち "1" のビットは "0" に、"0" のビットは "1" にする論理否定です。

命令	ディスティネーション	ソ                      ス		
		レ   ジ   ス   タ	メ   モ   リ	定 数
AND OR XOR	A	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	定数
CPL	A	A	—	—

アセンブリ言語ではディスティネーションはオペランドに書かない。

CPL ではソースも書かない。

**AND**

a	b	結果
0	0	0
0	1	0
1	0	0
1	1	1

(例) A レジスタ

ソース

A レジスタ  
(結果)

0	1	0	0	1	0	1	1
1	1	0	1	0	0	0	1
0	1	0	0	0	0	0	1

**OR**

a	b	結果
0	0	0
0	1	1
1	0	1
1	1	1

(例) A レジスタ

ソース

A レジスタ  
(結果)

0	1	0	0	1	0	1	1
1	1	0	1	0	0	0	1
1	1	0	1	1	0	1	1

**XOR**

a	b	結果
0	0	0
0	1	1
1	0	1
1	1	0

(例) A レジスタ

ソース

A レジスタ  
(結果)

0	1	0	0	1	0	1	1
1	1	0	1	0	0	0	1
1	0	0	1	1	0	1	0

**CPL**

前	結 果
0	1
1	0

(例) A レジスタ

A レジスタ  
(結果)

0	1	0	0	1	0	1	1
↓	↓	↓	↓	↓	↓	↓	↓
1	0	1	1	0	1	0	0

## 50 ビット操作命令

レジスタ、メモリ内のどこの1ビットでも、ビット操作命令で"0"か"1"にすることができ、また"0"か"1"かの判定をすることができます。コントロール的な応用にはよく使う命令で、Z-80の特徴の一つといえます。

セット[SET]命令は、ビットを"1"にする命令で、オペランドにはビットの位置とレジスタ名か、メモリのアドレスを示すレジスタHL、IX、IYのいずれかを指定します。ビットの位置は、2の $n$ 乗の $n$ で表わしますので、右端が0、左端が7となります。

リセット[RES]命令は、ビットを"0"にする命令です。オペランドはセット命令と同じです。

ビット[BIT]命令は、指定のビットを調べ、"0"であればFレジスタのゼロフラグ(Zフラグ)を"1"にして、"0"であったことを記憶しておきます。後に書かれた条件付ジャンプ命令で、"0"のときと"1"のときの飛び先番地を別々に指定してあれば、条件に応じた手順で仕事を進めます。オペランドはセット命令と同じです。

23A6H番地のメモリの右から3番目のビットを"1"にするときは、HLレジスタに23A6Hをロード命令で書き込んでから[SET 2, (HL)]を実行させ、"0"にするときは[RES 2, (HL)]、内容を判定するには[BIT 2, (HL)]となります。

命 令	ディスティネーション	ソ ー ス		
		レ ジ ス タ	メ モ リ	ビット
SET RES BIT	ソースと同じ	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	0-7

アセンブリ言語ではディスティネーションはオペランドに書かない。

HL

0326

のとき

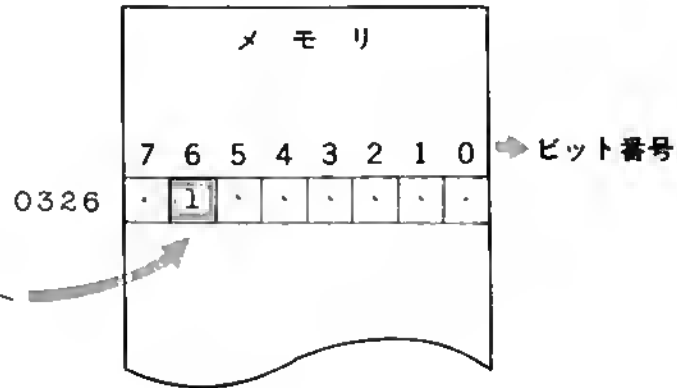
SET 6, (HL)

を実行すると

メモリの

0326H 番地の第6ビット

が "1" になる



SET "1" にする

RES "0" にする

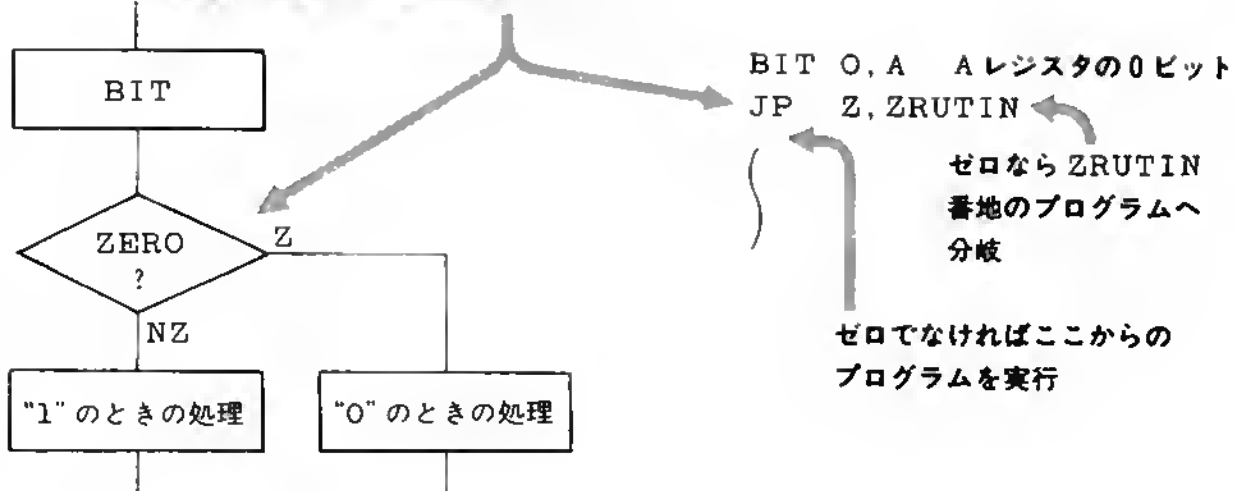
BIT 指定のビットが "0" が "1" によって

Fレジスタの Z フラグを "0" なら Z (ゼロ)

"1" なら NZ (ノンゼロ) を示すように設

定する

この場合、後にある判定命令 (ジャンプ命令) で Z フラグの状態を判定し以後の処理を分岐させる。



## 51 ローテート、シフト命令

**ローテート命令**は、ビットパターンを右か左に一つだけずらす命令です。はみ出したビットは最後に回り込んでつながります。このときキャリフラグを含めて回転するか、含めずに回転し、回り込んだビットと同じものをキャリフラグへ入れるかによって大別されます。ライト ローテート [RR]、レフト ローテート [RL]、ライト ローテート サーキュラ [RRC]、レフト ローテート サーキュラ [RLC] があり、特に Aレジスタに対する命令は、8080 系の CPU と互換性を持たせるために、1 バイト命令が別に用意されています。これ以外のローテートとシフトは Z-80 特有の機能です。

ライト (レフト) ローテートデジット [RRD], [RLD] 命令は、HL レジスタの内容で示されるメモリと A レジスタの下 4 ビットとの間で 4 ビット (1 デジット) を一まとめにして回転させる命令で、2 進数 10 進数の多桁演算によく使われるものです。

**シフト命令**は、はみ出したビットを切りすて、最後には "0" または左端ビットの値 (符号ビット) が入ります。シフト ライト アリスメチック (算術的右シフト) [SRA]、シフト レフト アリスメチック [SLA]、シフト ライト ロジカル [SRL] の三つです。算術的シフトは、ビットパターンを 2 進数とみなしたときシフトしても符号が変わらないよう配慮されているシフトです。左シフトは算術的 (アリスメチック) も論理的 (ロジカル) も同じですから一つしかありません。左に一つシフトすると、数値は 2 をかけた値になり、右にシフトすると 2 で割った値になります。

ビットパターンを 2 進数扱いでかけ算、割り算をするときは、シフトとローテートを使って高速のルーチンを組むことができます。

命令	ソース (デスティネーション)		動作
	レジスタ	メモリ	
RLCA	A*	—	
RLC	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
RLA	A*	—	
RL	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
RRCA	A*	—	
RRC	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
RRA	A*	—	
RR	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
RLD	A+(HL)*		
RRD			
SLA	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
SRA			
SRL			

\* 印はアセンブリ言語ではオペランドに記述しない

Cy はレジスタの C フラグ (キャリ)

SRA の 7 番ビットは変わらない

シフトでは Cy の元の値は失われる

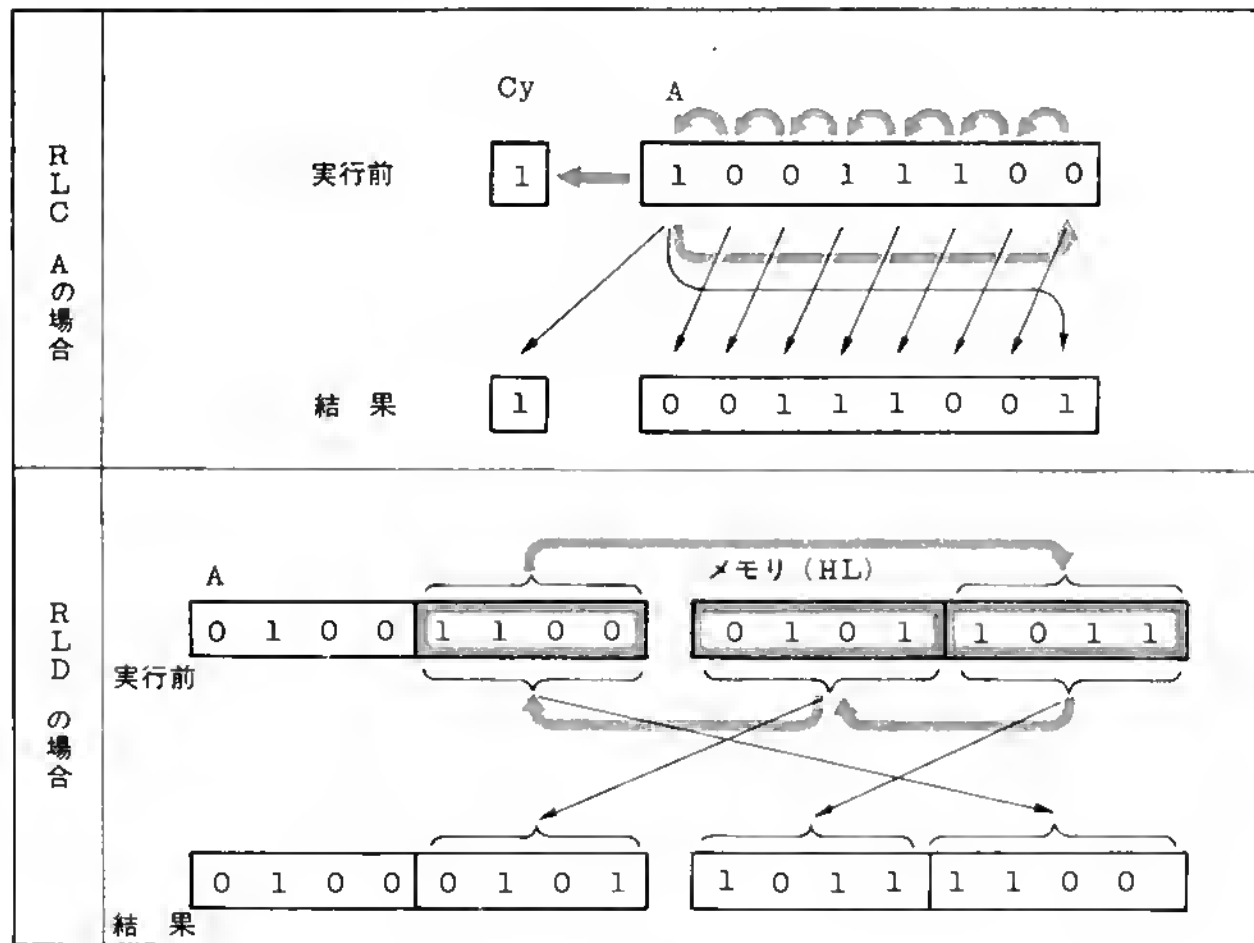
RRD, RLD では移動する 4 ビット単位の内容は変わらない

## 51 ローテート、シフト命令

(例)

Cy    A  
1    1 0 0 1 1 1 0 0    のときの結果は

命 令	Cy	A
RLC A	1	0 0 1 1 1 0 0 1
RL A	1	0 0 1 1 1 0 0 1
RRC A	0	0 1 0 0 1 1 1 0
RR A	0	1 1 0 0 1 1 1 0
SLA A	1	0 0 1 1 1 0 0 0
SRA A	0	1 1 0 0 1 1 1 0
SRL A	0	0 1 0 0 1 1 1 0





## シフト、ローテート を使ったプログラム例

(16ビットのかけ算サブルーチン)

メモリに図のように a と b の値が 2 進数で入っている。

$$a \times b = c$$

の演算をし、答 c を 32 ビットとして、メモリに格納する。

メモリのアドレス n は IX レジスタに入れられているとする。

```

MPX  LD    C, (IX+0)
      LD    A, (IX+1)
      LD    E, (IX+2)
      LD    D, (IX+3)
      LD    B, 16
      LD    HL, 0
      EXX
      LD    HL, 0
      LD    DE, 0
      EXX
LOOP SRL    A
      RR    C
      JR    NC, JNDT
      ADD   HL, DE
      EXX
      ADC   HL, DE
      EXX
JNDT SLA    E
      RL    D
      EXX
      RL    E
      RL    D
      EXX
      DJNZ LOOP
      LD    (IX+4), L
      LD    (IX+5), H
      EXX
      LD    (IX+6), L
      LD    (IX+7), H
      RET

```

n

メモリ	
a	(下 位)
a	(上 位)
b	(下 位)
b	(上 位)
c	(最下位)
c	(下 位)
c	(上 位)
c	(最上位)

## 52 ブロック転送、ブロックサーチ、ブロック入出力命令

メモリ内の複数バイトのブロックを別の番地へ移し変える場合、1バイトずつCPU内のレジスタへ読み込み、別の番地へ書き込む操作を必要バイト数になるまで繰り返します。また、メモリ内のブロックの中から指定のビットパターンと同じパターンを持つ1バイトを探し出すときも、1バイトずつAレジスタへ読み込んでコンペア〔CP〕命令を実行し、一致するまで繰り返します。入出力のときもメモリブロックの内容をポートへ次々に出力したり、次々にポートから読んだデータをメモリへ並べたりするときは繰り返しのプログラムを組まなければなりません。これを一つの命令で置き換えられるのが、この命令群です。

**ブロック転送命令**は、HLレジスタに元のアドレス、DEレジスタに転送先のアドレス、BCレジスタに転送バイト数をあらかじめ書いておき、次に転送命令を実行させますと、HLとDEを1ずつ進め、BCを1ずつ減らしていきます。

**ブロックサーチ命令**は、HLレジスタにアドレス、BCレジスタにブロックのバイト数を、比較すべきビットパターンをAレジスタに書いておきます。一致するとFレジスタのゼロフラグが、BCレジスタがゼロになる（一致するものがない）とオーバフローフラグ（Vフラグ）がリセットされますので、次の命令でフラグ判定をしなければなりません。

**ブロック入出力命令**は、HLにデータ格納のメモリアドレス、CレジスタにIOポートアドレス、Bレジスタにバイト数を書いてから実行させます。

これらの命令には、メモリブロックを先頭から扱うインクリメントグループと後から扱うデクリメントグループがあります。また、全データの操作が終る（サーチでは一致した場合も含む）までプログラムカウンタが変わらないで、自動的にループするリピートの機能を持つ命令もあります。転送、サーチ、入出力しながら、何かのデータ操作がない場合は、リピートは便利です。1バイト扱うごとにフェッチサイクルから始まります。

## ブロック転送

(HL←HL+1の意味は、HLの内容に1を加えるということ)

命 令	動 作		
LDI	HLの内容を番地とするメモリから DEの内容を番地とするメモリへ転送する	HL←HL+1 BC←BC-1	1バイトだけ転送して終る
LDIR		DE←DE+1	BC=0までくり返す
LDD		HL←HL-1 BC←BC-1	1バイトだけ転送して終る
LDDR		DE←DE-1	BC=0までくり返す

BC=0のときはVフラグがセットされる

## ブロックサーチ

命 令	動 作		
CPI	HLの内容を番地とするメモリの内容とAレジスタの内容を比較する	A-(HL)=0...等しい	HL←HL+1
CPIR		A-(HL)≠0...等しくない	BC←BC-1
CPD		等しいときZフラグをセット (ゼロの状態)	HL←HL-1
CPDR			BC←BC-1
			1バイトだけ調べて終る
			BC=0か A-(HL)=0まで
			1バイトだけ調べて終る
			BC=0か A-(HL)=0まで

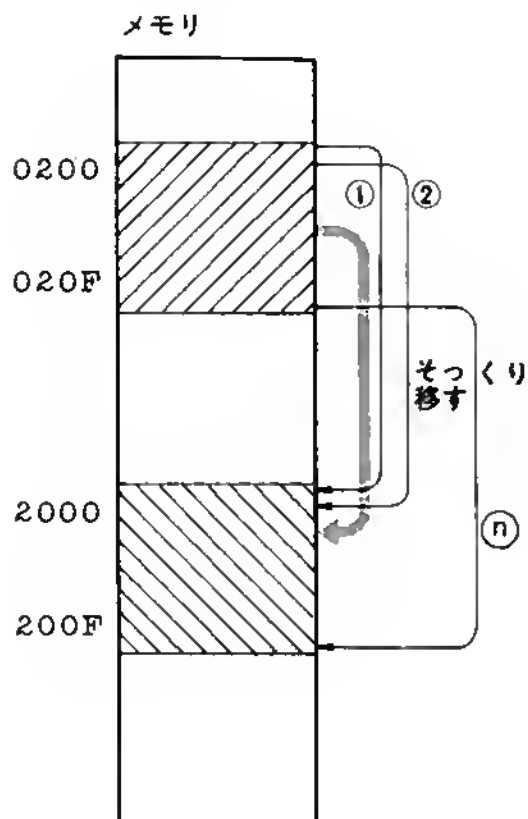
BC=0のときはVフラグ、A-(HL)=0のときはZフラグがセットされる

## ブロック入出力

命 令	動 作			注
INI	Cの内容を番地とするポートからHLの内容を番地とするメモリへ読み込む	HL←HL+1	1バイト入力して終る	Cは変わらない カウンタはBだけ (256まで)
INIR		B←B-1	B=0までくり返す	
IND		HL←HL-1	1バイト入力して終る	
INDR		B←B-1	B=0までくり返す	
OUTI	HLの内容を番地とするメモリの内容をCの内容を番地とするポートへ出力する	HL←HL+1	1バイト出力して終る	
OTIR		B←B-1	B=0までくり返す	
OUTD		HL←HL-1	1バイト出力して終る	
OTDR		B←B-1	B=0までくり返す	

B=0のときはZフラグがセットされる

## 52 ブロック転送, ブロックサーチ, ブロック入出力命令



```
LD HL, 0200H
```

```
LD DE, 2000H
```

```
LD BC, 0010H
```

```
LOOP LD A, (HL)
```

```
LD (DE), A
```

```
INC HL
```

```
INC DE
```

```
DEC BC
```

```
LD A, B
```

```
OR C
```

```
JP NZ, LOOP
```

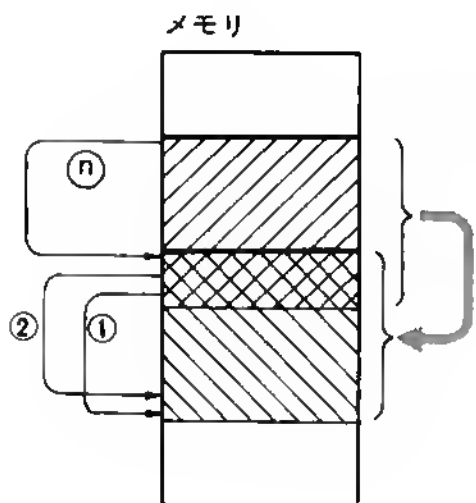
=LDI

=LDIR

B, C ゼロであることを調べる

→ ノンゼロなら LOOPへ

← ゼロなら次へ (終り)



転送前と後のエリアが

重なっていると、最初のほうから転送した場合これから転送すべき内容の所へ書いてしまい、内容が正しく残らない。

このようなときは最後のほう(下)から転送する  
= LDDR 命令を使う

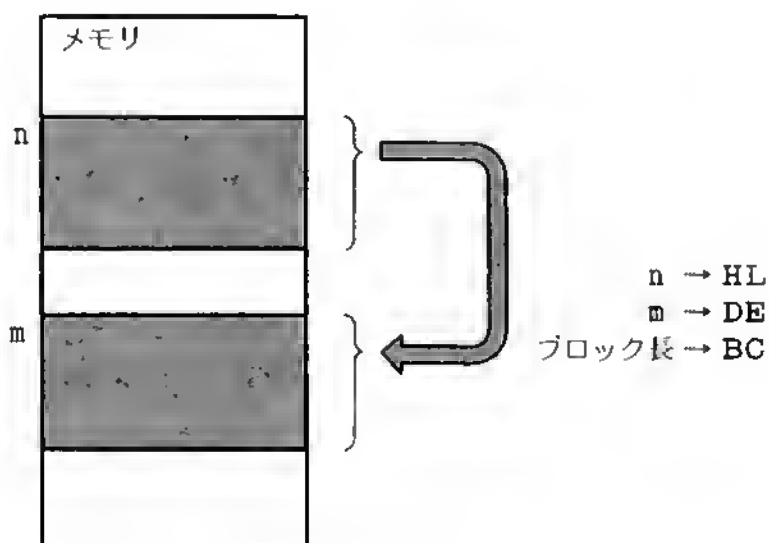
## ブロック転送命令を使ったプログラム例

(ブロック転送サブルーチン)

HLレジスタに転送前アドレス、DEレジスタに転送後アドレス、BCレジスタに転送するブロックのバイト数が入っているとする。転送前と転送後のエリアが重なっている場合を想定して、正しく転送できるように[LDIR]と[LDDR]を使い分ける、

```

MVE  PUSH  HL      ; HL を退避
      AND   A        ; キャリをゼロに
      SBC   HL, DE   ; キャリを変化
      POP   HL      ; HL を元にもどす
      JR    NC, IRR  ; 上から転送
      DEC   BC       ; BC - 1
      ADD   HL, BC   ; HL を変更
      EX    DE, HL
      ADD   HL, BC   ; DE を変更
      EX    DE, HL
      INC   BC       ; BC + 1
      LDDR          ; 下から転送
      RET
IRR   LDIR          ; 上から転送
      RET
  
```



## 53 ジャンプ命令

メモリに入れられたプログラムはゼロ番地から順次実行しますが、順序を入れ替えたり、条件によって別の番地のプログラムへ分岐したりするときに使います。

**ジャンプ [JP] 命令**は、条件を伴わない無条件ジャンプと、判定条件付ジャンプ命令があり、飛び先番地は絶対番地がつけられます。

判定のジャンプ命令は条件付ジャンプと呼ばれ、フラグレジスタの内容によって、ジャンプするかジャンプせずに次の命令へいくかの決定をします。この条件には、ゼロフラグ、キャリフラグ、パリティフラグ、サインフラグがそれぞれ“0”である、または“1”であるとき、ジャンプせよというものです。条件に合わないときは、そのジャンプ命令がないのと同様に次の命令を実行します。

**ジャンプ リラティブ [JR] 命令**は、ジャンプ先のアドレスを絶対値で持つのではなく、現在のプログラムカウンタ（自分自身のアドレス）からのへだたりで持っています。数値としては-128 から+127 バイトで、0 のときは次の命令の先頭番地です。アセンブリ言語で書くときは、飛び先番地のラベルを指定すれば、自動的に計算してくれます。ジャンプ リラティブは2 バイト命令ですが、ジャンプ命令（3 バイト）より実行時間がかかります。また条件判定は、ゼロフラグとキャリフラグについてのみしかできません。しかしジャンプ先までのへだたりが変わらなければジャンプ先がどこの番地になってもマシン語は同じになりますので、小さなプログラムモジュール内で使った場合、モジュールの配置は自由（リロケータブル）となります。

**ディクリメントジャンプノンゼロ [DJNZ] 命令**は、変った命令で、Bレジスタから1を引き、ゼロでなければジャンプリラティブと同じ方式で示された番地へジャンプするというものです。繰り返しに入る前にBレジスタへ必要回数を書き、繰り返し処理の終りにこの命令をおけば、終了しなければもう1回繰り返しの先頭へジャンプ、終了すればぬけ出て次へ、と分岐させることができます。回数はBレジスタが8ビットですから256回（Bレジスタにゼロを入れておくと256回で終る）までです。

## 無条件ジャンプ

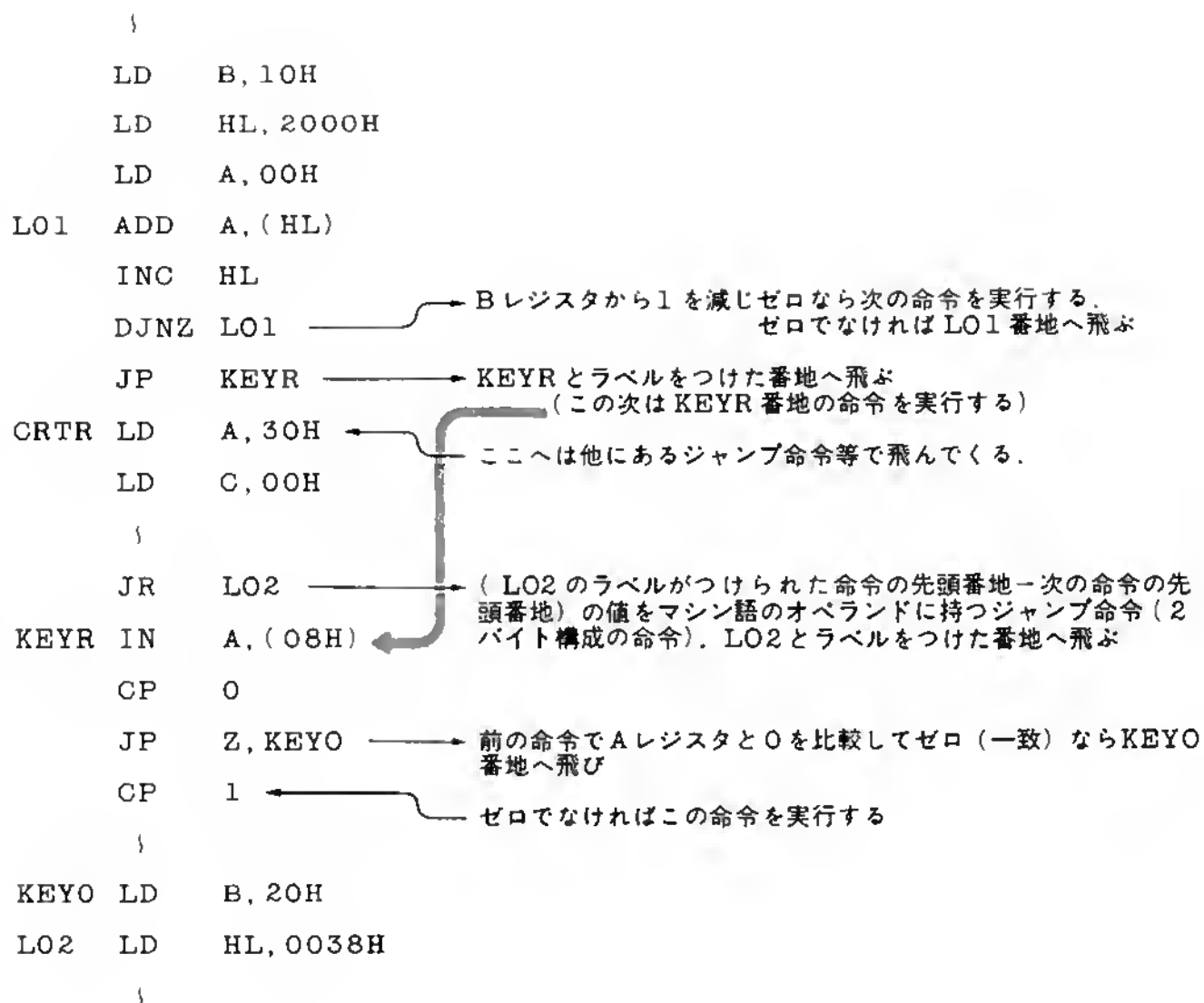
命 令	オペランド
JP nn	<p>飛び先のメモリにつけられた番地をそのままオペランドにつける(絶対番地)</p> <p>(例) JP <u>0258H</u> ⇒ マシン語では C3 <u>58</u> <u>02</u>  <small>下位 上位</small>  アセンブリ言語ではラベル名を書いてもよい</p> <p>オペランドには (HL) (IX) (IY) を使用してもよい。</p>
JR nn	<p>飛び先のメモリの番地までのディスプレイズメントをオペランドにつける(相対番地)</p> <p>(例) JR LO2 ⇒ LO2 が 0255H 番地で次の命令が 0250H 番地  とすると、マシン語では 18 <u>05</u></p>

## 条件ジャンプ

条件が合えばジャンプする、合わなければ次の命令へ

命 令		条 件
JP	JR	
JP Z, nn	JR Z, nn	Z フラグを調べゼロの状態 (Z フラグ=1) ならジャンプ
JP NZ, nn	JR NZ, nn	Z フラグを調べノンゼロの状態 (Z フラグ=0) ならジャンプ
JP C, nn	JR C, nn	C フラグを調べキャリ有り (C フラグ=1) ならジャンプ
JP NC, nn	JR NC, nn	C フラグを調べキャリなし (C フラグ=0) ならジャンプ
JP PE, nn	—	P/V フラグを調べイーブン(偶数) (P/V フラグ=1) ならジャンプ
JP PO, nn	—	P/V フラグを調べオッド(奇数) (P/V フラグ=0) ならジャンプ
JP M, nn	—	S フラグを調べマイナス (S フラグ=1) ならジャンプ
JP P, nn	—	S フラグを調べプラス (S フラグ=0) ならジャンプ
—	DJNZ nn	B ← B - 1 B ≠ 0 ならジャンプ

注 JR のオペランドはアセンブラによってはラベル名ではなく、相対値を書くものもある。





## よいプログラムとは

### 1. 実行速度が速いこと

用途によっては、速度が問題にならない場合もあるが、まずほとんどの場合、全システムの効率に大きく影響する。

### 2. メモリをくわないこと

プログラム自体の長さは、無駄を省いた最少限でなければならない。実行速度を上げるためにも重要である。また、専用システムではメモリ数を減らし、部品点数を引き下げることにより得られるメリットは多い、ただし、以下の項目を犠牲にしないよう心がけなければならない。

データ格納用メモリは、情報処理（事務計算）のような用途で多用されるが、フロッピディスク等の外部記憶を有効に利用し、システム効率を上げるような設計が望ましい。

### 3. デバッグ（修正）しやすいこと

いかに有能なプログラマでもでき上ったプログラムが1回で動作するのは希である、また、でき上ってから何年もたってからバグ（誤り）が発見されることがある、すみやかに対処できるよう配慮しなければならない、次項とも共通するが、コメントを豊富に、機能をモジュール化し分散させること、フラグの使用は最少限に、サブルーチンからジャンプでメインルーチンへもどるなど論外である。

### 4. 改造しやすいこと

新しくプログラムを作るとき、前に作ったプログラムの改造で対応できれば、こんな楽なことはない、設計変更されそうな点を予測すること、

### 5. 速く作ること

プログラムの原価は、ほとんど人件費につきる。速く、しかも後で応用のきくプログラムを作ることが、プログラマの使命である。

## 5.4 コール、リスタート、リターン命令(サブルーチン)

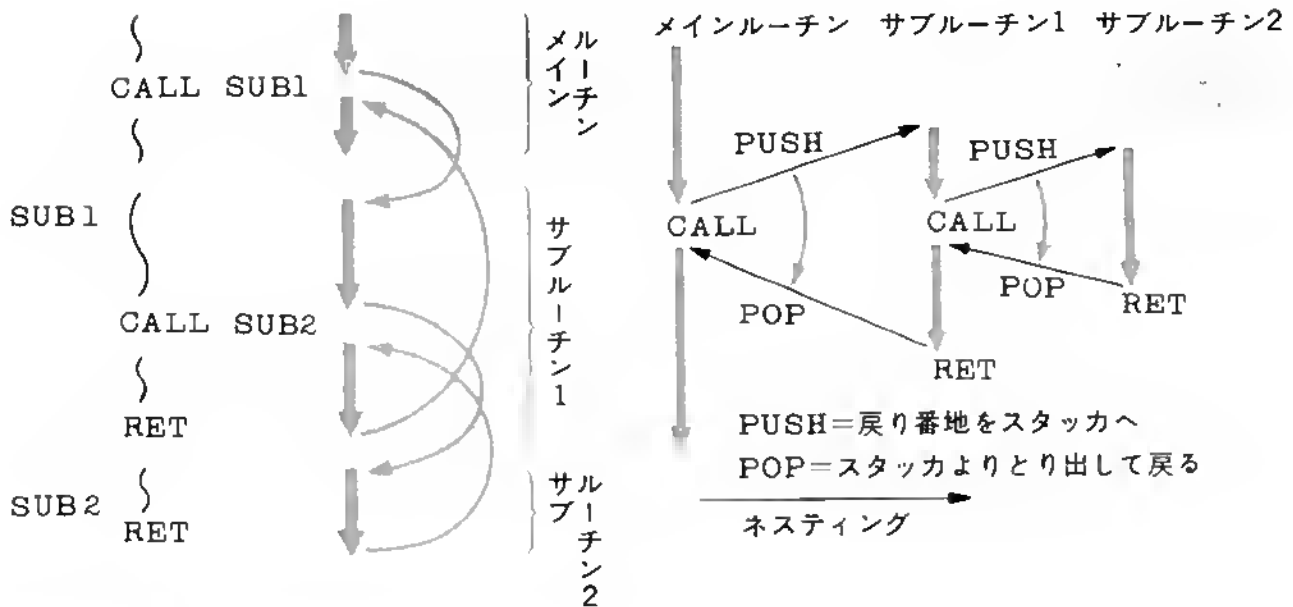
**コール [CALL] 命令**は、ジャンプ命令と似ていますが、ジャンプするとき次の命令の番地をスタッカへ自動的にプッシュ [PUSH] します。そしてリターン [RET] 命令があるとスタッカからポップ [POP] した値をプログラムカウンタへ入れて元の流れへ戻ります。コールで呼び出される番地からリターン命令までを**サブルーチン**と呼び、プログラムの中のどこからでも呼び出して実行させることができます。コール命令は2バイトのジャンプ先の絶対番地を持った3バイト命令です。コール命令にもジャンプ命令と同じ条件を付けることができます。条件を満たしてないときは、コール命令がないのと同様に、ジャンプせずに次の命令を実行します。

**リスタート [RST] 命令**は、特定の8個の番地へのコール命令に相当します。1バイト命令なので、多用されるサブルーチンをコールするときに便利な命令です。

リターン命令であってもノン マスカブル 割り込み [NMI] からのリターンは [RETN] を使います。他の割り込みに対する割り込み 禁止 状態を解除するためです。また、割り込み (INT) からのリターンは、Z-80 のペリフェラルのデージェチェーンによる優先順位決定機能を使うときは、リターン フロム インタラプト [RETI] 命令を使って、ペリフェラルに対して割り込み処理の完了を知らせてやる必要があります。これは割り込み処理ルーチンの中でサブルーチンコールを行なう場合もあり、サブルーチンからのリターンで割り込み処理完了になってしまうようにするためです。

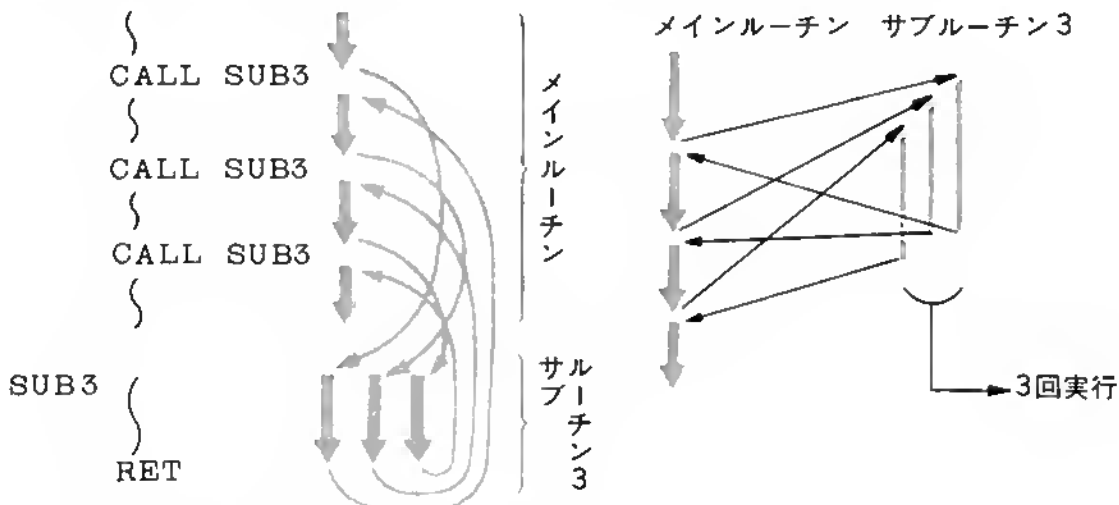
サブルーチンの使い方は、プログラムの良否にかかわる重要なテクニックです。単に同じ手続きをあちこちで何回も使うから、記述の手間を省くというだけでなく、プログラムの機能のブロック化、あるいは単機能モジュールのブラックボックス化に役立ちます。デバッグ済のサブルーチンは内容を知らなくても使い方を覚えておけば、いつでも呼び出せます。モジュールとしてたくさん作っておけば、次からはモジュールの組み合わせを変えるだけで新しいプログラムができていくわけです。

## 54 コール、リスタート、リターン命令(サブルーチン)



(例1) メインルーチンからサブルーチン1をコールし、さらにサブルーチン2をコールする

(例2) メインルーチンのあちこちから1つのサブルーチン3をコールする



命 令	オ ペ ラ ン ド	
CALL nn	nnは絶対番地	無条件コール
CALL cc, nn	nnは絶対番地 ccは条件	条件付コール 前項のJPと同じ条件
RET	オペランドはない	無条件リターン
RET cc	ccは条件	条件付リターン 前項のJPと同じ条件
RST n	nは 00H 08H 10H 18H 20H 28H 30H 38H のいずれかを選ぶ n=08Hとすると、0008H へのコール命令(CALL 0008H) と同じ働きをする、ただし RST nは <b>1バイト命令</b>	

## 55

## レジスタとフラグ変化

レジスタは六つのフラグにより構成されます。フラグはそれぞれが意味を持ち、演算命令やローテート命令、入出力命令などの実行結果によって変化します。

**キャリ フラグ (Cフラグ)** は、最上位のビットからの桁上り、桁がりによりあるいはローテート、シフトで変化します。

**ゼロ フラグ (Zフラグ)** は、実行結果がゼロになったとき "1" にセットされます。

**パリティ/オーバフローフラグ (P/V フラグ)** は、パリティ (ビットパターンの "1" の数が奇数か偶数か) と、符号付演算のオーバフローを兼ねています。パリティは、奇数なら "0"、偶数なら "1"、オーバフローは、オーバフローして本来正の数になるべき結果が負の数になってしまったとき "1"、正しい結果のとき "0" になります。

**サイン フラグ (Sフラグ)** は、符号付演算の符号ビット (左端ビット) と同じになります。負数であれば "1"、正数であれば "0" です。

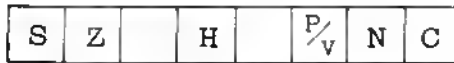
**ハーフ キャリ (Hフラグ)** は、下位 4 ビットに対するキャリ フラグです。

これらのフラグのうち、サブトラクト フラグとハーフ キャリフラグは、ジャンプ命令では判定できませんが、2 進化 10 進数を扱うデシマル アジャスト アキュムレータ [DAA] 命令を実行するとき参照されます。

フラグは命令により変化する、変化しない、不定になる、各場合があります。

コンプリメント キャリ フラグ [CCF] 命令は、他のレジスタに変化を与えずにキャリフラグのビットパターンを反転させる命令です。セット キャリ フラグ [SCF] 命令はキャリフラグを "1" にする命令です。なお "0" にしたいときは [AND A] か [OR A] を使います。

## Fレジスタ



## → キャリフラグ

加算、減算で桁上り、桁下りがあったとき1になる。  
ローテート、シフトでは、各ビットにつれて移動する。

## → サブトラクトフラグ

(DAA 命令実行用=プログラムでは操作できない)  
命令によりセットされるかリセットされるか決まっている。  
加算系のとき0、減算系のとき1になる。

## → P/V フラグ

命令により P:パリティフラグと V:オーバフローフラグに  
使いわける。

## パリティフラグ

論理演算、Cレジスタ間接ポート指定の入力命令等で、  
ディスティネーションに乗ったデータのビットパターン  
の1の数が偶数(イーブン)のとき1、奇数(オッド)  
のとき0になる。

## オーバフローフラグ

演算の結果が符号付算術演算の結果としてオーバフロー  
したとき1、すなわち8ビット演算では-128~+127、  
16ビット演算では-32768~+32767以外を表現する  
必要が出たときセットされる。したがって、正数のみを対  
象とした符号なし演算、(8ビットでは0~255、16ビットで  
は0~65535の範囲で定義した演算)のときは無視してよい。  
この場合はCフラグがオーバフローを表わしている。

## → このビットは使用していない

## → ハーフキャリフラグ

(DAA 命令実行用=プログラムでは操作できない)  
8ビットの演算で下4ビットからの桁上げ、桁下りがあっ  
たとき1になる。

## → このビットは使用していない。

## → ゼロフラグ

実行の結果がゼロのとき1、ゼロ以外のとき0になる。

## → サインフラグ

実行の結果が符号付算術演算の結果として負の数になっ  
たとき(この場合最上位ビットが1になる)1になる。

## 56

## 2進化10進数と10進補正命令

CPU内の算術演算は、メモリやレジスタに記憶されたビットパターンを2進数として演算します。8ビットレジスタでは0～255まで、256以上を表わすと桁上げをしますが、レジスタの内容は0と256が同じパターンになります。多くの桁を扱う場合は、桁上げ操作や入出力を簡単にするため、2進化10進数（バイナリーコードデシマル＝BCD）を用います。

4ビットのパターンは16種類0～Fまでありますが、このうち0～9までを有効とし、A～Fについては上位桁の4ビットへ1繰上げ0～5にします。このようにすると4ビット＝1桁の10進数表現ができます。10桁の10進数を使いたいときはメモリに5バイトのエリアを確保し、2桁ずつ演算するのです。

しかし、演算はBCDで表現されていても、2進数で計算しますので、結果としてA～Fが出現してしまいます。ここでデシマルアジャストアキュムレータ〔DAA〕命令を実行すると、いまの計算が加算か減算か、ハーフキャリは出ているか、などフラグレジスタの状態によって必要な桁上げ、桁がり処理をして、BCDに戻してくれます。BCDはデータエリアが多少大きくなりますが、入出力はほとんどの場合10進数が要求されますので、変換の手間が省け、多桁演算がやりやすいなど特徴が多く、よく使われる手法です。

ビットパターン	名前	BCD 表現
0 0 0 0	0	ここまでを 10進数に対応 させる 1010 は 上の位へ桁上げし 0000 に戻る
0 0 0 1	1	
0 0 1 0	2	
0 0 1 1	3	
0 1 0 0	4	
0 1 0 1	5	
0 1 1 0	6	
0 1 1 1	7	
1 0 0 0	8	
1 0 0 1	9	
1 0 1 0	A	使わない
1 0 1 1	B	
1 1 0 0	C	
1 1 0 1	D	
1 1 1 0	E	
1 1 1 1	F	

8+6 を例にとって

加算すると

10進数では

$8+6=14$  であるが

2桁のBCDでは

$0000\ 1000 = 08$

$+ ) 0000\ 0110 = 06$

$0000\ 1110 = 0E$

となり、下4ビットは使わない値になってしまう、

ここでDAA命令を実行すると

$0000\ 1110$



$0001\ 0100$

↓

1

↓

4

になおしてくれる

この補正のし方は加減算、

桁あふれ等により異なるが、

Fレジスタの働きで間違えることはない

### BCD の表し方

$1001\ 0110$

9

6

↑

↑

10の位

1の位

2桁の10進数

(4ビットを1デジット(桁)と呼ぶのは  
ここからきている)

### 2進数

$1001\ 0110$  は

16進数 になおすと (名前と呼ぶと)

96 である、これを

10進数になおすと

$9 \times 16^1 + 6 \times 16^0 = 150$  である

### これをBCDと定義すると

$1001\ 0110$  は

16進読み にすると

96 (きゅうろく)

10進数にしても

96 (きゅうじゅうろく) である

コンピュータの中で数値を表わすには

1. 2進数として扱う
2. BCDとして扱う
3. 指数表現を取り入れる

等のやり方がある

1. は入出力のたびに10進数との変換が必要、プログラムは簡単でスピードも早い、桁数が自由にならない。
2. は桁数が自由になり、入出力、プログラムは簡単、メモリにむだが多い。
3. は、桁数が多い場合に有効、プログラムは複雑で演算時間がかかる。

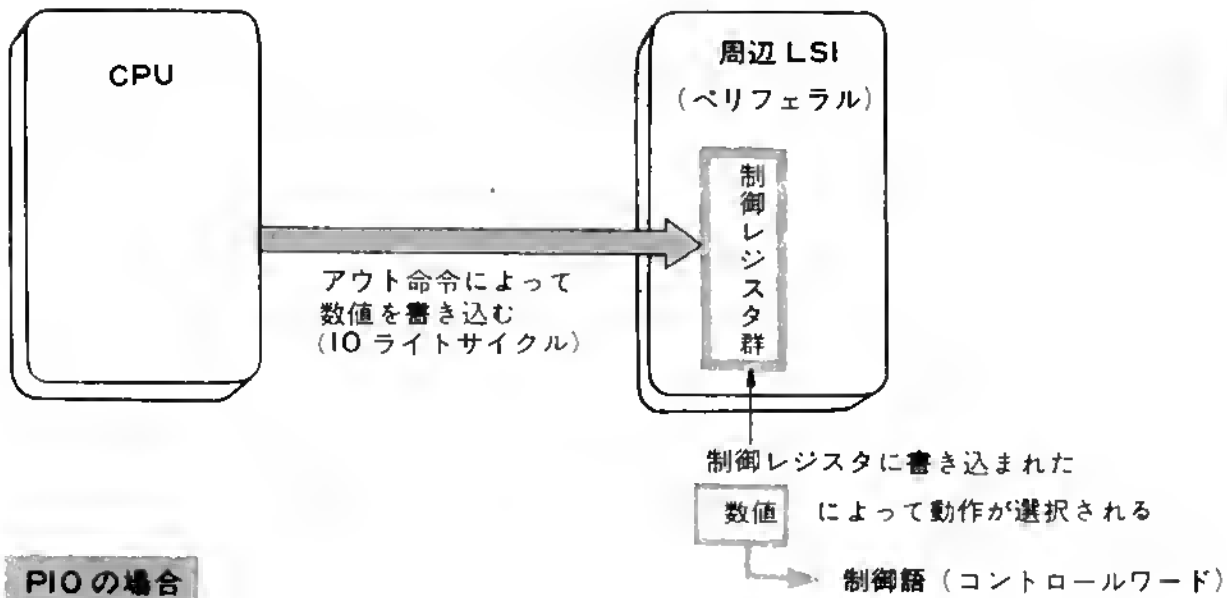
## 57 ペリフェラルのプログラミング

ペリフェラルを使うときは、データの入出力を行なう前に、ペリフェラルの内部レジスタへ情報を書き込んで、目的にあった動作をするように設定しなければなりません。機能がよいほど選択の幅が広がりますので、たくさんの情報を与えなければなりません。情報は次々に書き込みます。情報の中に何の情報かを示すビットが設けられている場合もありますが、書き込む順序によって、意味が違ってしまふことがあります。マニュアルに記載された順序を守れば安全です。

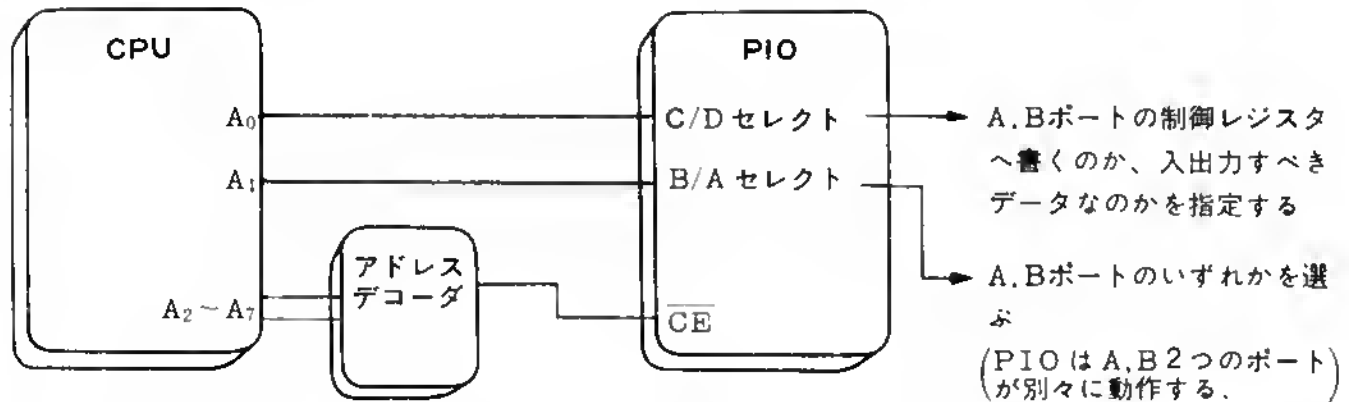
ペリフェラルのレジスタへ情報を書き込むには、そのペリフェラルのつながっている IO ポートへ出力命令を出します。PIO と SIO には、C/D セレクト信号端子があり、“1” のとき C すなわちコントロールワード、“0” のとき D すなわちデータ、とセレクトしますので、制御情報を書き込むときはこの端子を“1”にして出力命令を出します。実際にはこの端子をアドレスバスのいずれかのビットへつなぎますと、制御ワードとデータとは別の IO ポートアドレスに配置されることになります。CTC のチャンネルセレクトや PIO の B/A ポートセレクトも同じ考え方で対処します。

一度書き込んだ制御情報は、リセットされるか、または新たに書き込まれるまで有効です。また、特別な場合を除いて IN 命令で読み出すことはできません。





## PIO の場合

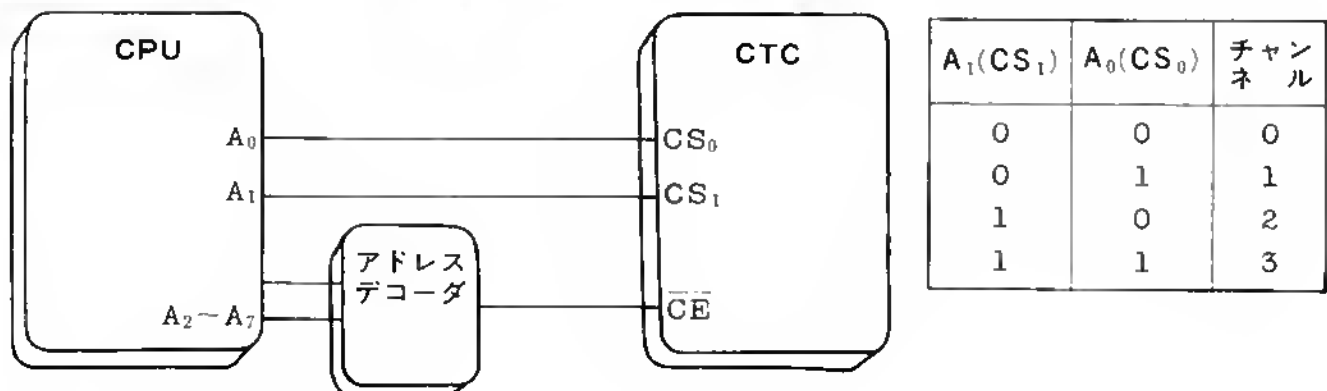


A <sub>7</sub> ~ A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	番 地	ポート切換え	データ/制御語切換え
×	0	0	×0, ×4, ×8, ×C のどれか	B/Aセレクト が"0"だから Aポート	C/Dセレクトが"0"だからデータ
×	0	1	×1, ×5, ×9, ×D のどれか		// "1" // 制御語
×	1	0	×2, ×6, ×A, ×E のどれか	B/Aセレクト が"1"だから Bポート	// "0" // データ
×	1	1	×3, ×7, ×B, ×F のどれか		// "1" // 制御語

×はアドレスデコーダによってきまる

## CTC の場合

(CTC に対する入出力は制御レジスタにのみ適応する)  
(CS<sub>0</sub>, CS<sub>1</sub> は内部の 4つのチャンネルを選択する)



## 5.8 PIO モード 0 の動作

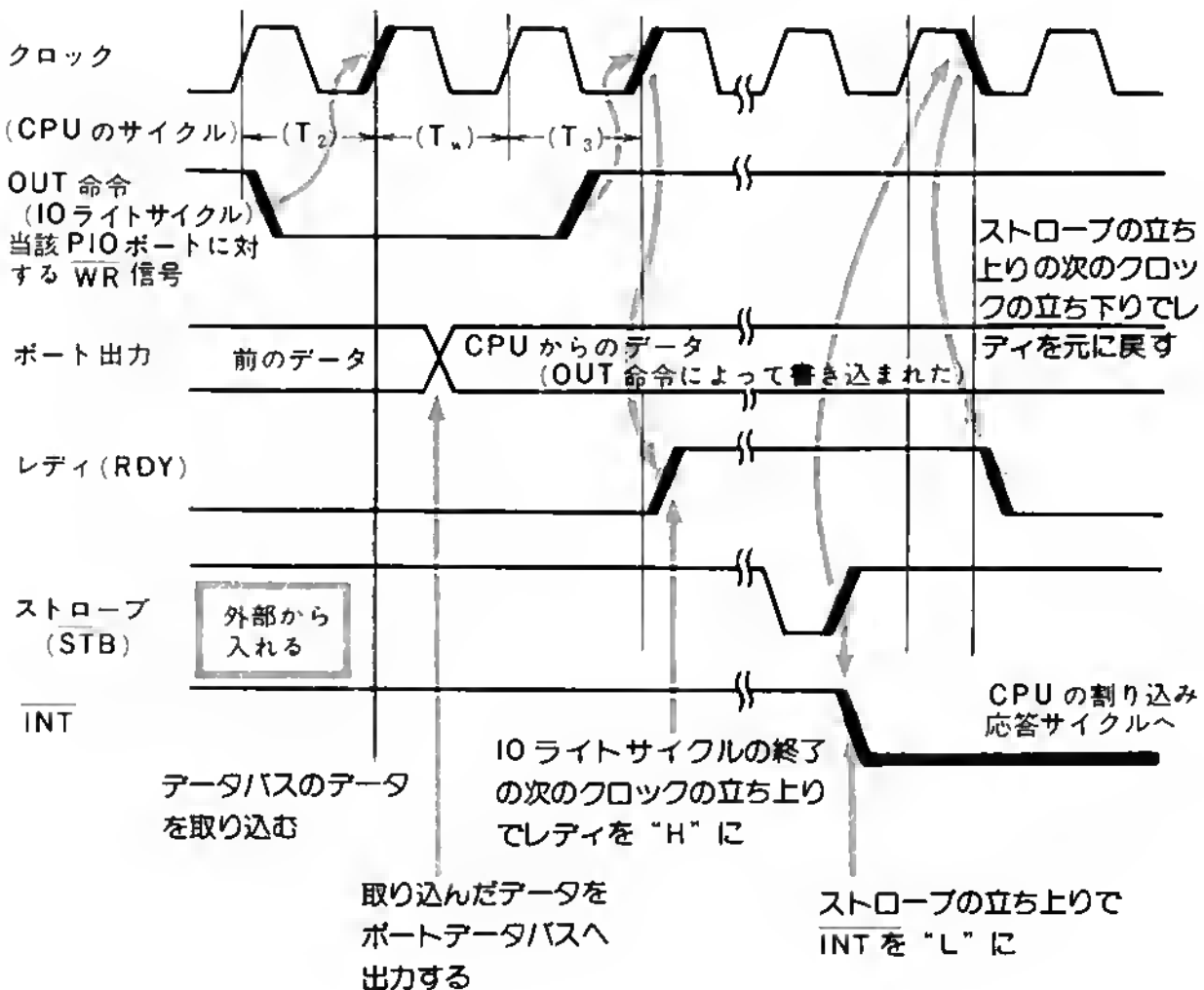
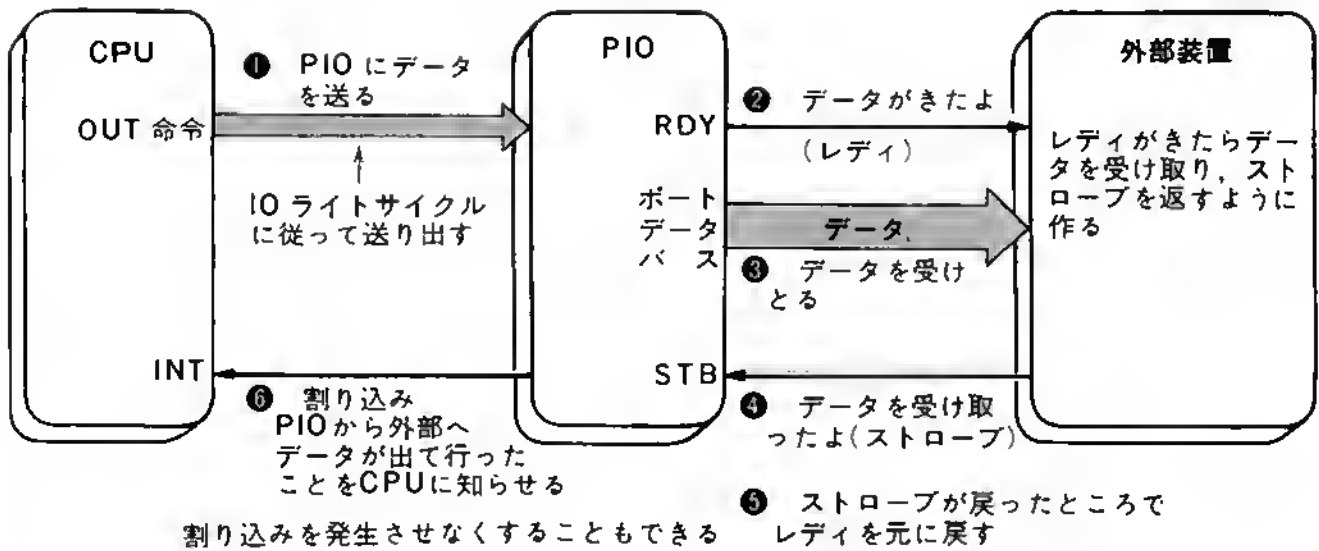
モード 0 は出力モードです。8 ビットの入出力線に出力 [OUT] 命令で出されたデータバスの状態がそのまま出てきます。データバスの内容が変わっても次に出力命令がくるまで出力線の内容はそのままです。

CPU からの出力動作では、IO リクエスト ( $\overline{\text{IORQ}}$ ) とライト ( $\overline{\text{WR}}$ ) 信号が "L" になりますが、PIO にはライト信号の入力端子がありません。これは ( $\overline{\text{IORQ}}$ ) が "L" で、リード ( $\overline{\text{RD}}$ ) が "H" で、チップイネーブル ( $\overline{\text{CE}}$ ) が "L" で、 $\overline{\text{C/D}}$  がデータすなわち "L" のとき PIO 内部でライト信号を作り出しているからです。

出力命令が出されるとデータバスから PIO の出力レジスタへ信号が入ります。内部のライト信号が立ち上ると次のクロックの立ち下りで、すでに出力線に出力レジスタの内容が乗り確定しています。同時にレディ端子からレディ (RDY) 信号が出されます。外部装置はこのレディ信号が "H" になったことにより出力線のデータを取り込み、終わったらストローブ ( $\overline{\text{STB}}$ ) 端子を "L" にして応答します。レディ信号は、このストローブの立ち上りの次のクロックの立ち下りで "L" に戻り、次のデータの出力命令を待ちます。割り込みがイネーブルにプログラムされていて、優先順位の高い割り込みがかかっていなければ、ストローブの立ち上りで割り込み ( $\overline{\text{INT}}$ ) 出力は "L" になって CPU に割り込みをかけることができます。次の出力命令があるまで出力線は変わりません。

レディ信号が "H" のとき、すなわち出力命令に対してストローブが返っていないときにさらに出力命令を出すと、レディ信号は一度 "L" に戻ってすぐ "H" になります。レディ信号は正論理信号ですから "H" アクティブです。

レディとストローブの信号線をハンドシェーク線と呼びます。外部装置と CPU は PIO のハンドシェーク線を通じて互いに確認し合いながら (同期をとりながら) データのやりとりを行なうわけです。

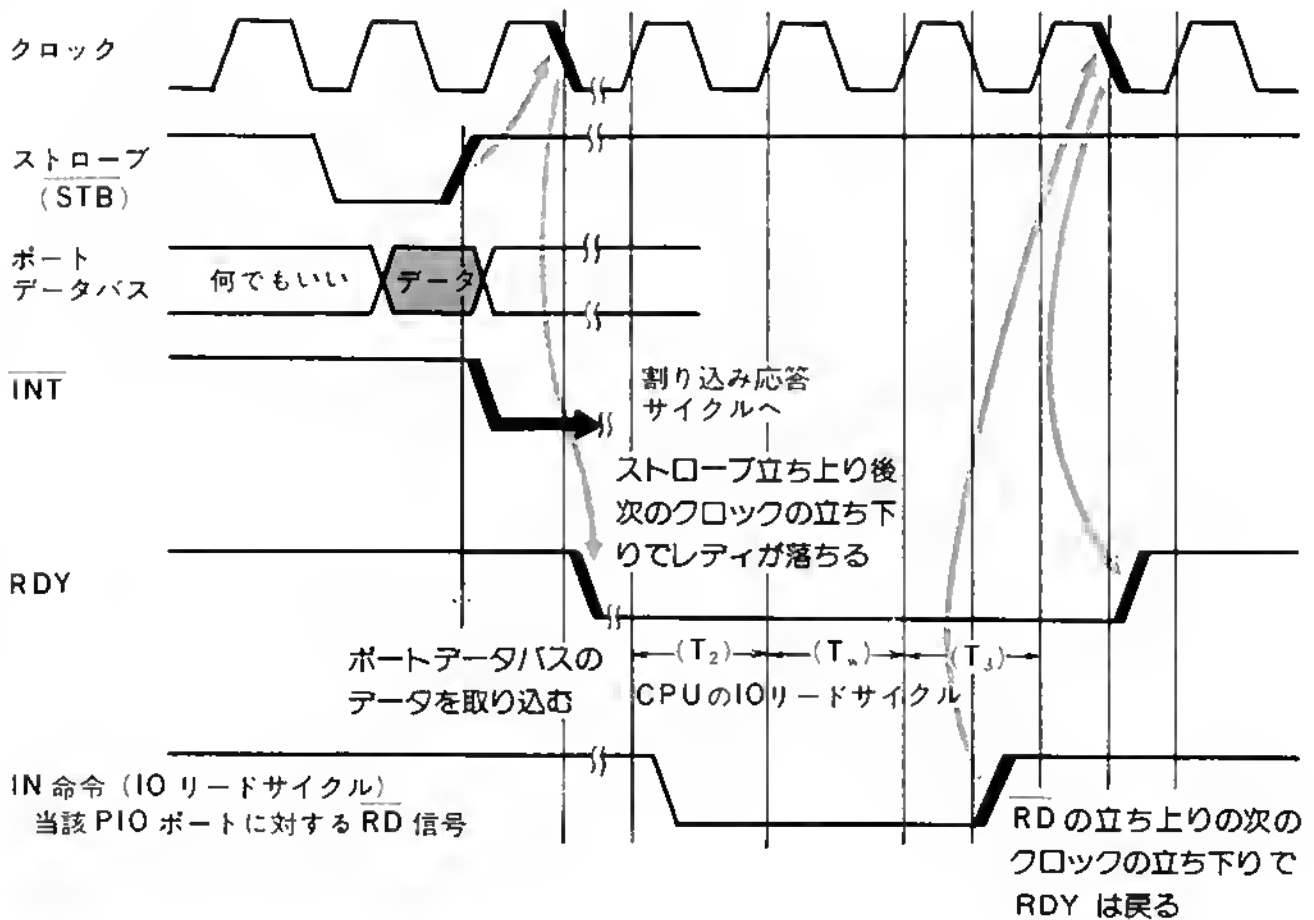
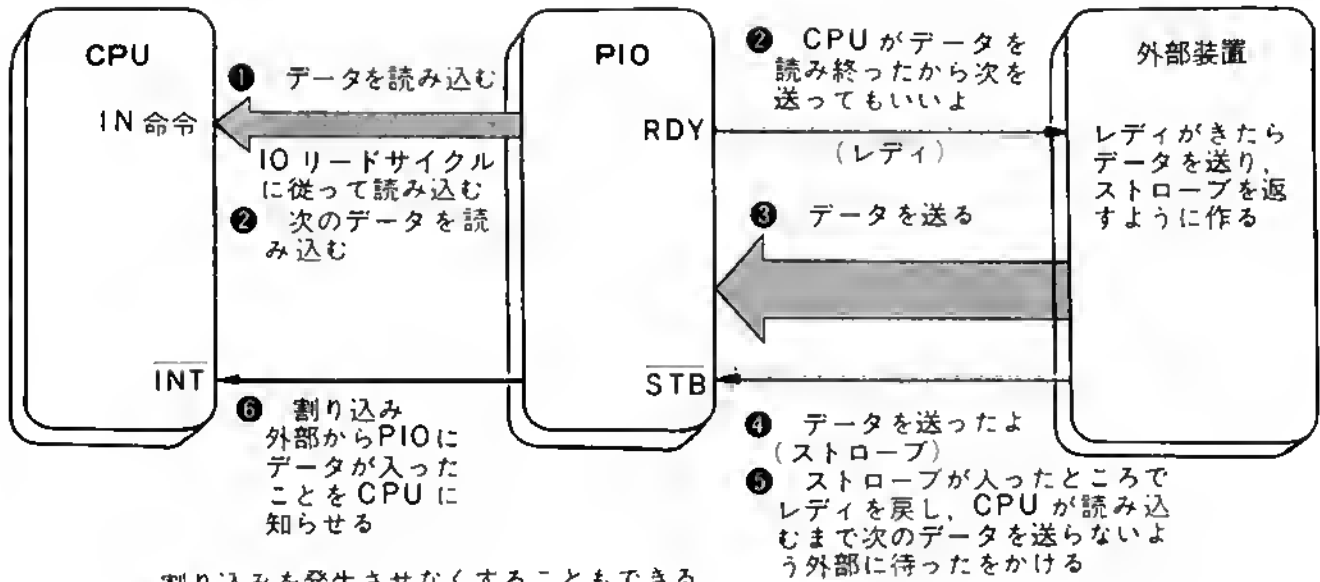


## 59 PIOモード1の動作

モード1は入力モードです。外部から入ってきた8ビットの信号がポートの入出力線に与えられ、CPUからの入力〔IN〕命令でデータバスを通じてCPUの内部レジスタへ取り込みます。

レディ端子は、リセットがかかると“L”になっています。CPUからの入力命令で“H”になって受け入れ可能を外部へ知らせます。この一発目の入力命令で読み込んだデータは無意味です。外部装置からレディが“H”のとき、入出力線にデータを与え、続いてストローブを“L”にするとストローブの立ち上りでPIOは入出力線から内部の入力レジスタへデータを取り込みます。次のクロックの立ち下りでレディを“L”にして外部から次のデータが入ることを禁止し、同時に割り込み可の状態であれば、CPUに割り込みをかけます。割り込み処理ルーチンで、入力〔IN〕命令を実行し、このデータをCPUの内部レジスタへ読み込みますと、リード（RD）信号の立ち上りと次のクロックの立ち下りでレディを“H”に戻し、次のデータ入力に備えます。

モード1では1個のダミーの入力命令が必要です。プログラミングが終り外部からの信号を受け入れてもよい状態になったら〔IN〕命令を実行してレディ信号を出し外部装置に受け入れ可を知らせて下さい。レディ信号が受け入れ状態を示していないときに、入力線にデータを乗せてストローブ信号を与えても、入力レジスタ上へデータを書き込むことはできません。しかし何らかの方法でCPUの入力命令の実行が遅れないようにしないと、データが失われるおそれがあります。



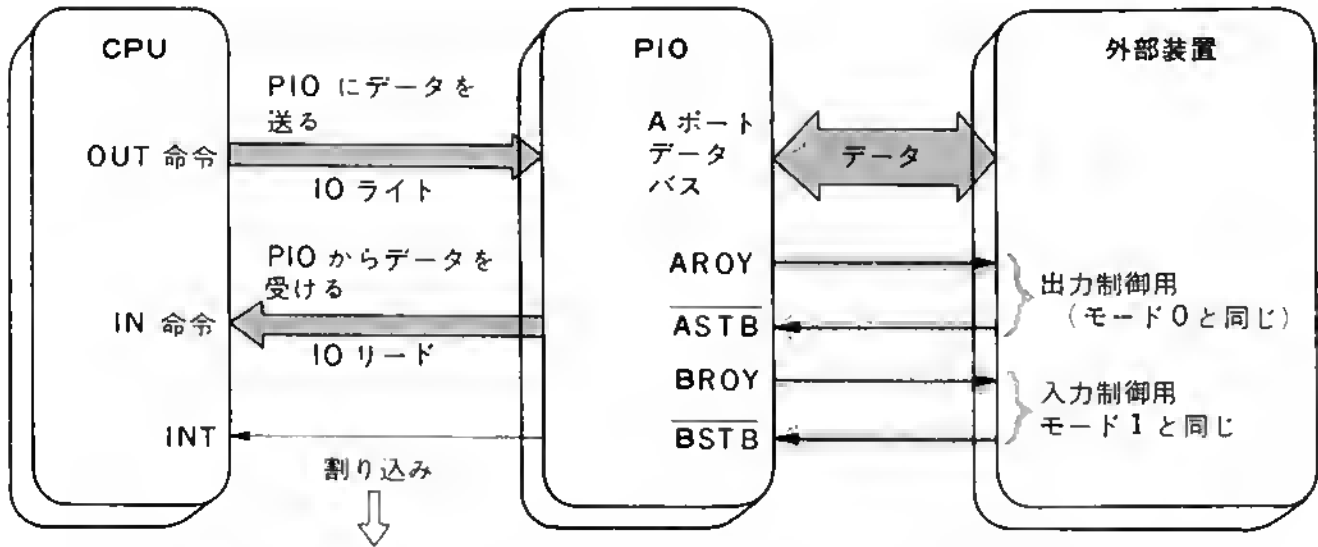
## 60

## P10 モード 2 の動作

モード 2 は A ポートだけについて可能な、入出力モードです。8 本のデータ線は入出力共用になります。ハンドシェーク線は A ポートが出力制御用、B ポートが入力制御用になります。したがって B ポートはハンドシェーク線を必要としないモード 3 に設定しなければなりません。

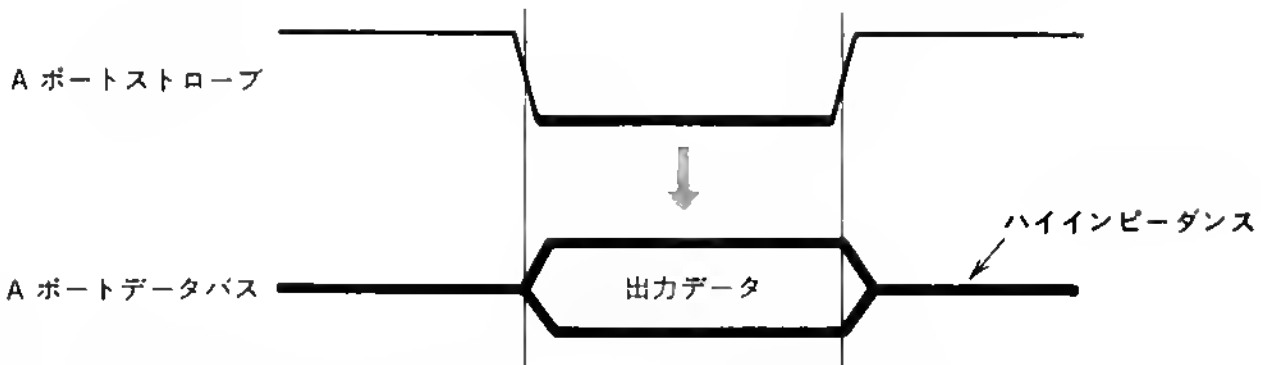
出力命令によるデータ出力動作はモード 0 と同じですが、モード 0 では出力線に常にデータが乗せられているのに対し、モード 2 のときは A ポートのストロープ (ASTB) が "L" になっている間だけ、多少遅れて乗せられてきます。外部装置はこのストロープを立ち上げるタイミングでデータを読み取ればよいのです。入力命令によるデータ入力動作は、B ポートのハンドシェーク線を使う場合はモード 1 と同じです。

割り込みをかけるタイミングも、モード 0、モード 1 と同じですが、入力動作では B ポートのハンドシェーク線を使うため、B ポートに書き込まれたベクトルが送出されます。したがって入力、出力各動作での割り込みを別々のベクトルで制御することができますが、B ポートをビットモードで使用する場合は、本来の B ポートの動作で発生する割り込みに対するベクトルと、A ポートの入力動作で発生するベクトルが同じ値になってしまいます。いずれかの割り込み機能を使用しないようにするか、処理プログラムの中で、どちらの割り込みかを判断しなければなりません。



入力動作と出力動作ではベクトルが異なる → 異なるように設定しておくこと

Bポートデータバスはモード3で使う。または使わない。ただしモード3に設定しておくこと



Aポートストロブが“L”のときだけ出力データがポートデータバスに現われる。したがってこのときは外部からデータを与えてはならない。他は、モード0、モード1と同じ。

## 6.1 PIO モード 3 の動作

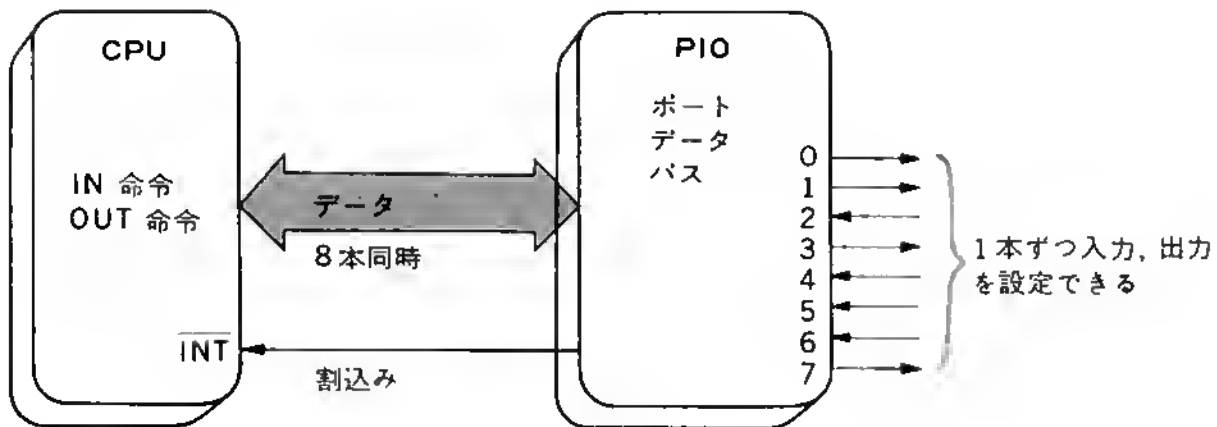
モード 3 はハンドシェイク線を使用しないで、非同期で入力出力を行いません。ビット単位に入出力の設定ができ、割り込みは入力に指定したビットが指定した状態になったときに入ります。

出力命令で送られたデータはモード 0 と同じタイミングで出力に指定されたビットに乗せられます。入力命令で読み込むと、リード ( $\overline{RD}$ ) 信号が立ち下る直前の入力に指定されたビットの状態が CPU に読み込まれます。出力に指定されたビットはそれ以前に出力命令で送り出したままが読み込まれます。

入力に指定したビットは、さらに割り込みに関係させるか否かのマスク指定をすることができます。マスク指定で割り込み要因のモニタビットに指定したビットが全部そろったとき (AND 条件)、いずれか一つが入ったとき (OR 条件) を選択でき、入力線を正論理 ("H" アクティブ) としてとらえるか負論理 ("L" アクティブ) としてとらえるかの選択もできます。

コントローラとしての用途では、PIO をモード 3 で使うことが多いと考えられます。周期的に CPU は入力ビットの状態を検査し、対応した出力信号を出力ビットへ乗せるようなプログラムを組みますが、入力ビットを読み込む周期はプログラムの長さで決まります。もっと早い対応が必要なシステムでは、割り込みを使うことにより解決できます。ただし割り込みによるシステムの応答は、外部装置で起こり得る最悪の条件のときを想定して設計しないと、CPU が追いつかなくなることもあります。時間計算から許容ステップ数を出し、その範囲内のプログラムを作るのですが、システムクロック周波数の決定とも合わせて慎重に検討しなければなりません。





#### 割り込み発生条件、設定

1. 入力ビットのうちどれを判断の対象とするか (マスク)
2. 入力は“H”アクティブか“L”アクティブか (論理)
3. AND か OR か (AND/OR)

たとえば、上の例で

- ・ビット 4 と 6 が共にアクティブになったとき割り込みを発生させたい
- ・入力は普通は“H”になって信号があるときだけ“L”になる (“L”アクティブ)

とすると

- ・マスク指定はビット 4 と 6
- ・論理は負論理
- ・AND/OR は AND と指定する

具体的には (次項参照)

ベクトル	× × × × × × × 0	⇒	× ×
モード設定	1 1 0 0 1 1 1 1	⇒	C F
ビット指定	0 1 1 1 0 1 0 0	⇒	7 4
	← 7ビット目      ← 0ビット目		
割り込み制御語	1 1 0 1 0 1 1 1	⇒	D 7
マスク指定	1 0 1 0 1 1 1 1	⇒	A F

↓  
これを順次PIOの  
制御レジスタ群へ  
OUTする

## 62 PIO のプログラミング

### 割り込みベクトル

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

→ "0" であることにより割り込みベクトルを意味する

→ 割り込み処理ルーチンのアドレステーブルの番地（下位）

### モード設定

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

→ "1111" であることによってモード設定語を意味する

→ 無意味（何でもよい）

→ モード

7	6
---	---

00 = モード0（出力モード）

01 = モード1（入力モード）

10 = モード2（双方向モード）

11 = モード3（ビットモード）

**ビット指定**    モード設定でモード3を指定したときは続けてこの指定をします。

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

→ ポートの入出力線のビット位置に対応したビットを、

入力ビットにするとき "1"

出力ビットにするとき "0" に指定する

制御語の書き込みはAポート、Bポートを別々に行なって下さい。

A、B混在させると、動作しない場合があります。順序もここに書かれているとおりなら問題ありません。

割り込みを使わない場合は、ベクトルの書き込みは省略できます。

## 割り込み制御語

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

マスク → "0111" であることにより割り込み制御語を意味する  
 モード 3 で割り込みに関係する入力ビットがある場合、続けてマスク指定を行なう、このとき "1"  
 割り込みを使わないとき、マスク指定はしなくてよいので、このときは "0"

H/L → モード 3 で割り込み発生要因となる入力ビットが "H" アクティブ(正論理)なら "1", "L" アクティブ(負論理)なら "0"

AND/OR → モード 3 で、割り込み発生要因となる入力ビットのすべてがアクティブなときに割り込み発生する (AND) なら "1"

いずれか一つ以上アクティブなとき発生 (OR) なら "0"

割り込み → 割り込みを発生させるとき "1", 発生させないとき "0"

## マスク指定

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

割り込み制御語のビット 4 が "1" のときは続けてこのマスク指定をする、ポートの入力線のビット位置に対応するビットを "1" にすると、割り込み発生とは無関係な入力になり、"0" にしたビットのみが割り込み発生要因となる

## 割り込み制御語

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

"0011" であるとき、割り込み発生の有無のみを変更する制御語を意味する

無意味 (以前の指定を変えない)

割り込みを発生させるとき "1", 発生させないとき "0"

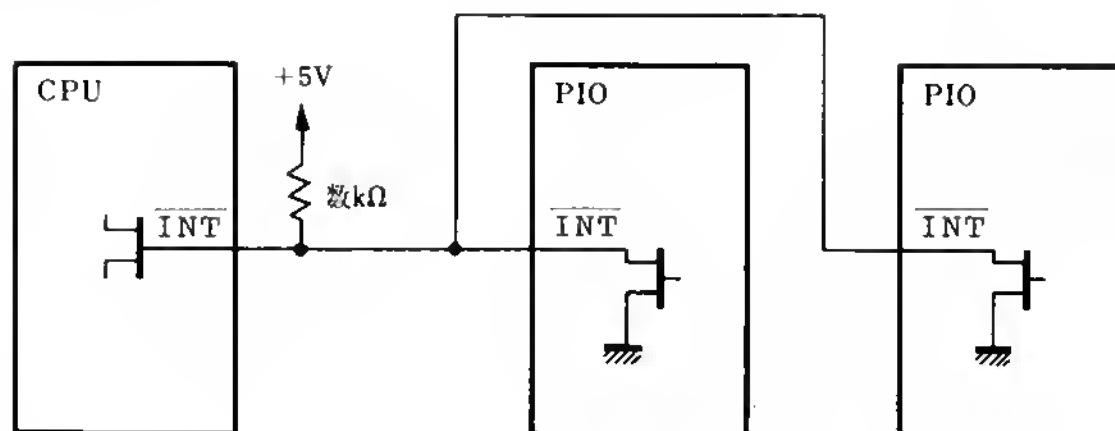
## 6.3 PIO のプログラム例

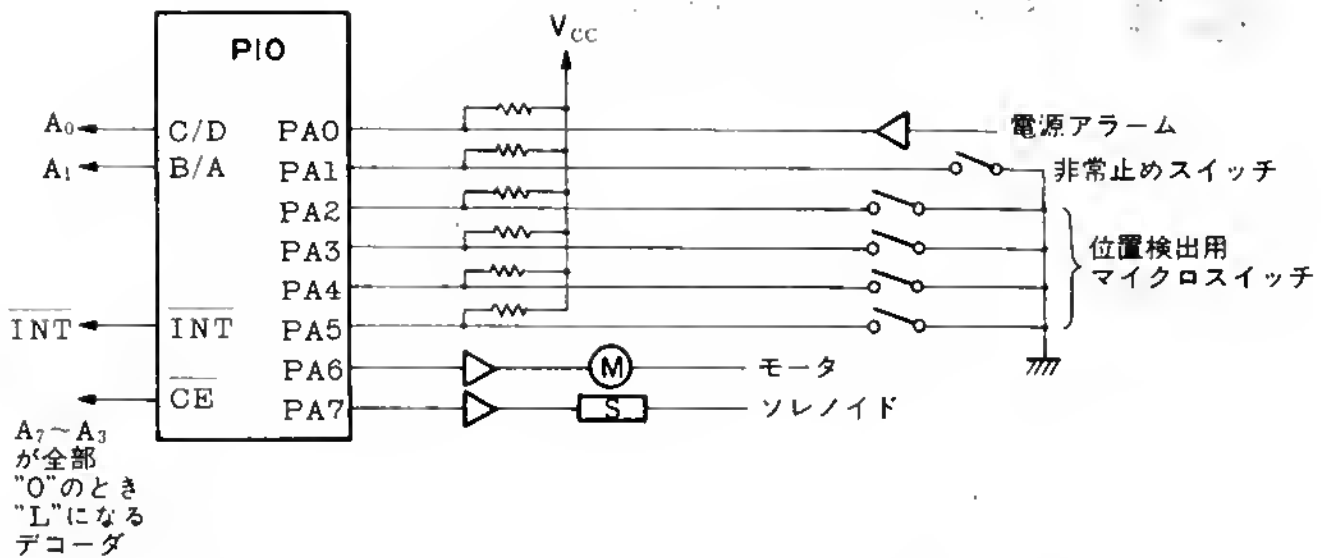
PIO をモード 3 で使用する場合の一例を考えてみます。PIO は IO ポートの 00H~03H に配置されているものとします。A ポートの 0~5 ビットを入力とし、他を出力ビットにします。電源異常信号と非常停止信号が共に "L" アクティブで 0 と 1 ビットにつながっています。処理に速度が要求されますので、割り込みをモード 2 で使用します。他の入力位置検出のマイクロスイッチの信号で出力はすべてモータやソレノイドを駆動します。出力を "H" にすると動き、"L" にすると止まります。応答速度は問題にならないとします。

まず 0000H 番地からのルーチンで PIO を設定します。設定の最後で CPU の割り込み受付を可能にします。次に入力ビットを読み込み、その条件に従った制御情報を A レジスタにととのえ出力ビットへ出力します。

割り込みが入ると、すべての出力を "L" にして動作を止めます。CPU はホルトに入り、新たにリセットがかかるのを待ちます。

PIO に限らず、ペリフェラルの  $\overline{\text{INT}}$  の出力端子はオープンドレインになっています。したがって、+5V へ抵抗でプルアップしておかなければなりません (ワイアードオア)。





```

; CONTROL                                コメント行
ORG 0000H                                ; このプログラムをゼロ番地から配置
LD SP, 0000H                             ; スタックポインタイニシャライズ
IM 2                                     ; 割り込みモード
LD A, 01H                                ; Iレジスタセット
LD I, A
LD A, 00H                                ; ベクトル
OUT (01H), A
LD A, 0CFH                               ; PIO モード設定
OUT (01H), A
LD A, 3FH                                ; ビット指定
OUT (01H), A
LD A, 97H                                ; 割り込み制御語
OUT (01H), A
LD A, 0FCH                               ; マスク指定
OUT (01H), A
EI                                         ; 割り込み許可
JOBS )                                   ; 仕事を始める

IN A, (00H)                               ; 入力ビットを読む
OUT (00H), A                             ; 出力をコントロールする
JP JOBS                                  ; 次の仕事を始める

ORG 0100H
DEFW INTR                                ; 割り込み処理ルーチンの先頭番地定義
; 割り込み処理ルーチン
INTR LD A, 0H                            ; モータを止める
OUT (00H), A
HALT                                     ; CPU を止める。リセット信号が入るまで停止
END                                     ; プログラム終了

```

PIO のプログラミング

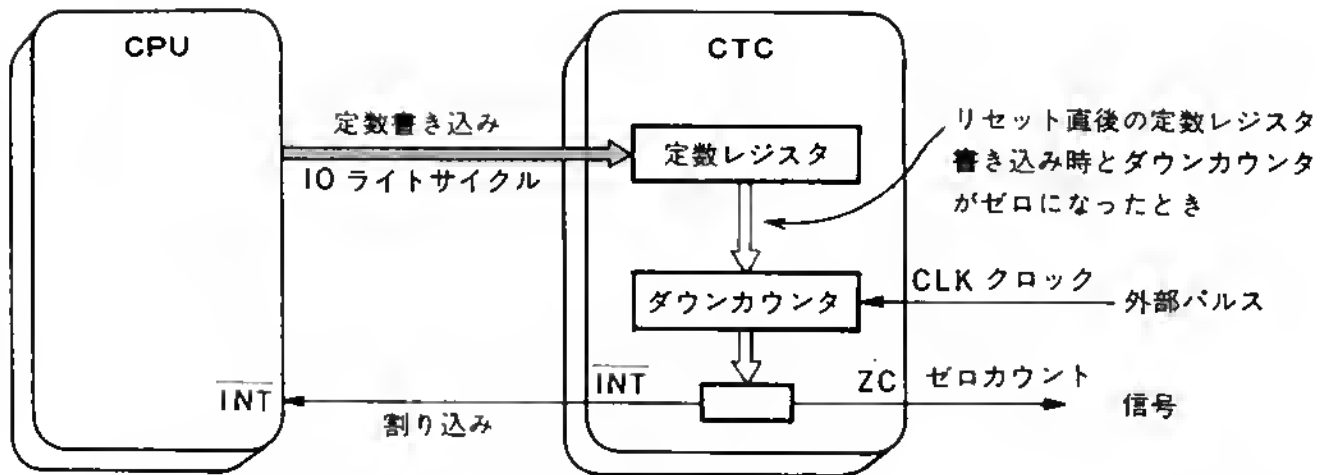
## 64 CTC カウンタモード

CTC のクロック/トリガ (CLK/TRG) 端子に与えられるパルスをカウントします。あらかじめ設定された数を 1 個のパルスで 1 ずつ減らしてゼロになると、次のクロックの立ち上りでゼロカウント/タイムアウト (ZC/TO) 端子を 1.5 クロックの間 "H" にし、割り込みを発生します。

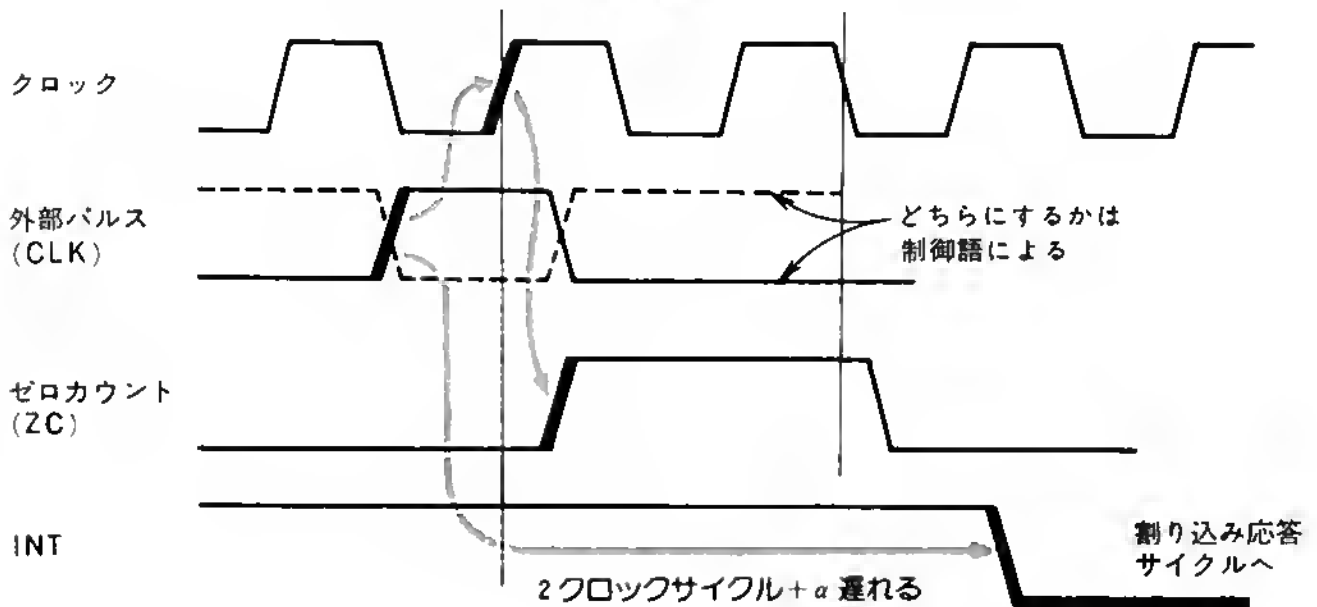
カウントするパルスはシステムクロックの 2 倍以上の周期で、"H" と "L" の時間はおのあの 120 ns 以上必要です。パルスの立ち上りまたは立ち下リエッジ (指定できる) がくると、次のクロックの立ち上りでダウンカウンタを 1 減じます。ダウンカウンタがゼロになるとゼロカウント信号を出し割り込みをかけます。同時に定数レジスタに設定されている数値をダウンカウンタへ書き込み、次のカウントに備えます。カウント途中で定数レジスタに新しい数値を書き込むと、次のゼロカウント時から有効になります。定数レジスタからダウンカウンタへの書き込みは、リセット状態から最初に定数レジスタが設定されたときと、ダウンカウンタがゼロになったときです。定数レジスタは 8 ビットですから 1 ~ 256 の値が設定できます。0 を設定したときは 256 を意味します。

割り込みベクトルレジスタは CTC 1 個 = 4 チャンネル分に 1 個しかありません。ベクトルの設定はチャンネル 0 に書き込みます。割り込みが発生すると要求するチャンネルによって、ビット 1 とビット 2 が決まったパターンに修飾されて送り出されてきます。優先順位はチャンネル 0 が一番高く順にチャンネル 3 が最後位です。

制御語や定数の書き込み読み出しのときのチャンネル指定はチャンネルセレクト ( $CS_0$ ) と ( $CS_1$ ) を切り替えて行ないます。アドレスバスの 2 本 ( $A_0$  と  $A_1$  がよく使われる) を接続すれば CPU 側からは四つのチャンネルが一連のポートアドレスに配列されます。PIO の C/D セレクト B/A セレクトと同じ考え方です。



ゼロカウントになった場合



## 65 CTC タイマモード

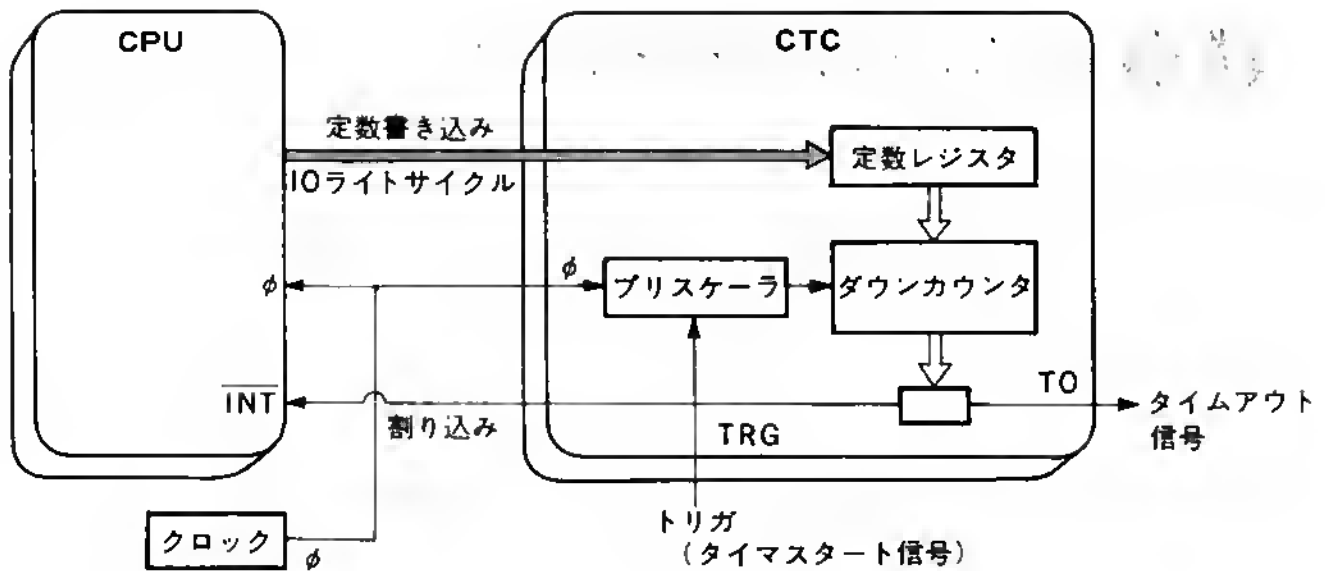
カウンタモードでは、不定期的なパルスをカウントしますが、決まった周期を持ったパルスをカウントすることにより時間経過を知ることができます。これがタイマモードの動作の基本です。カウントするパルスはプリスケアラで  $1/16$  か  $1/256$  に分周されたシステムクロックを使います。システムクロックの周期と分周数と、定数レジスタに書かれた数値の積がゼロカウント/タイムアウト (ZC/TO) 端子へ信号が出てくる時間周期となります。

タイマの起動は、トリガと自動が選択できます。トリガ起動の場合は、クロック/トリガ端子の立ち上りか立ち下りのエッジ (選択できる) から 2 回目のクロックの立ち上りでスタートします。自動の場合は、時間定数を書き込む出力命令の次の命令サイクルと同時にスタートします。

割り込みの発生と、定数の再設定についてはカウンタモードと同じです。

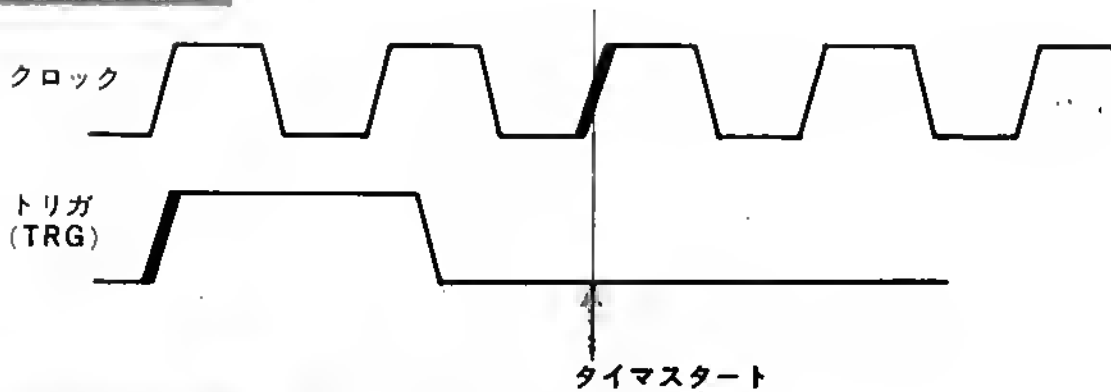
CTC はカウント中に読み出しても差しつかえありません。ダウンカウンタの現在値を知ることができます。ゼロカウント/タイムアウトを検知するのに ZC/TO 信号も割り込みも使わず、ダウンカウンタの値を読み出し、ゼロかどうか調べる方法が考えられますが、ダウンカウンタがゼロになり、次に定数レジスタの値が再びダウンカウンタへ書き込まれるまで、2 クロックしかないので、この間に CPU が読み出すとは限らないため、不確実で実用できません。



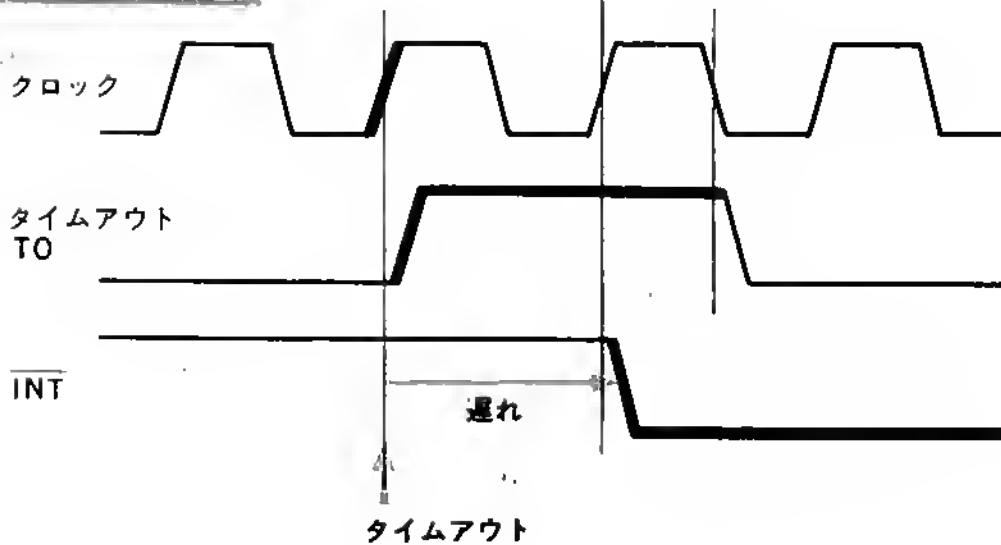


前項ではφは省略してあるが、  
カウンタモードでもクロックは  
与えなければならない

#### トリガ起動の場合



#### タイムアウトの場合

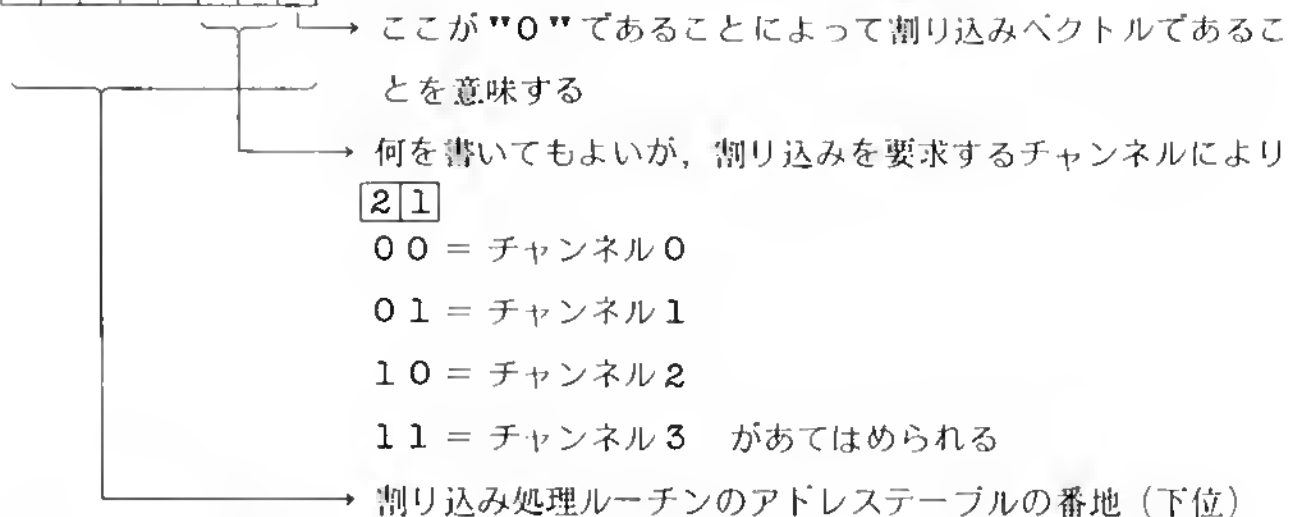


## 66 CTC のプログラミング

### 割り込みベクトル

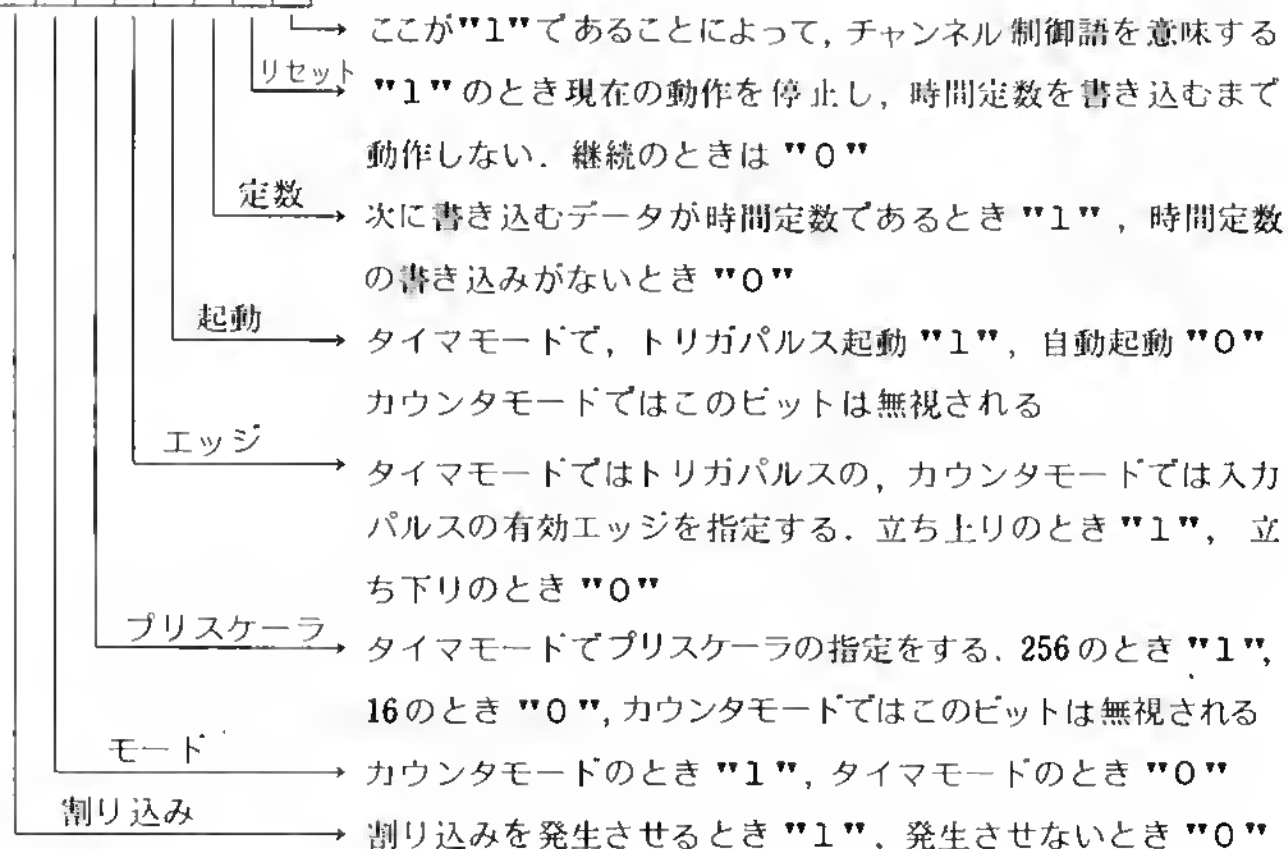
(チャンネル 0 に対してのみ書き込むことができます)

7 6 5 4 3 2 1 0



### チャンネル制御語 (各チャンネルごとに書き込みます)

7 6 5 4 3 2 1 0



## 時間定数

7	6	5	4	3	2	1	0
$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

→ チャンネル制御語のビット 2 が "1" のとき続けて書き込む。各ビットは 2 進数でゼロカウントになるまでの数値を表現している。ゼロのときは 256 として扱われる

## ファンイン ファンアウト

Z-80 ファミリは E-D-MOS プロセスという製法でできております。TTL のようなバイポーラとは異なり、規格の決め方にも相違があります。

入力端子は、0.8 V 以下で "L"、2.0 V 以上で "H" になり、電流は  $0 \sim V_{CC}$  V に対して  $10 \mu A$  以下しか流れません。ファンインとしてはごく小さいものです。

出力端子は、"L" が 1.8 mA 流れるとき 0.4 V 以下、"H" が  $250 \mu A$  流れ出して 2.4 V 以上となっており、ファンアウトとしてはノーマルタイプの TTL 1 個分、LS タイプなら 4 個までドライブできる能力を持っていることとなります。応用機器の必要ノイズ裕度や信頼性に応じてチェックしてから使用して下さい。

## 参 考 文 献

シャープ：Z-80 テクニカルマニュアル，エレクトロニクスダイジェスト社

渡部弘之訳：コンピュータ・データ通信技術，CQ 出版

森亮一監修：Z-80 マイクロコンピュータ，丸善

森下 巖：マイクロコンピュータ入門，昭晃堂

大川善邦：マイコン入門，コロナ社

平松啓二，森本淳堯：マイコン入門心得帖，オーム社

平松啓二，森本淳堯：続マイコン入門心得帖，オーム社

小牧常松，大條 廣：図解マイコンの使い方，オーム社

小牧常松，大原茂之：図解マイコンのためのアセンブラ入門，オーム社

平松啓二，斎藤 剛：図解マイコンのインタフェース，オーム社

矢田光治：図解マイコンの基礎知識，オーム社

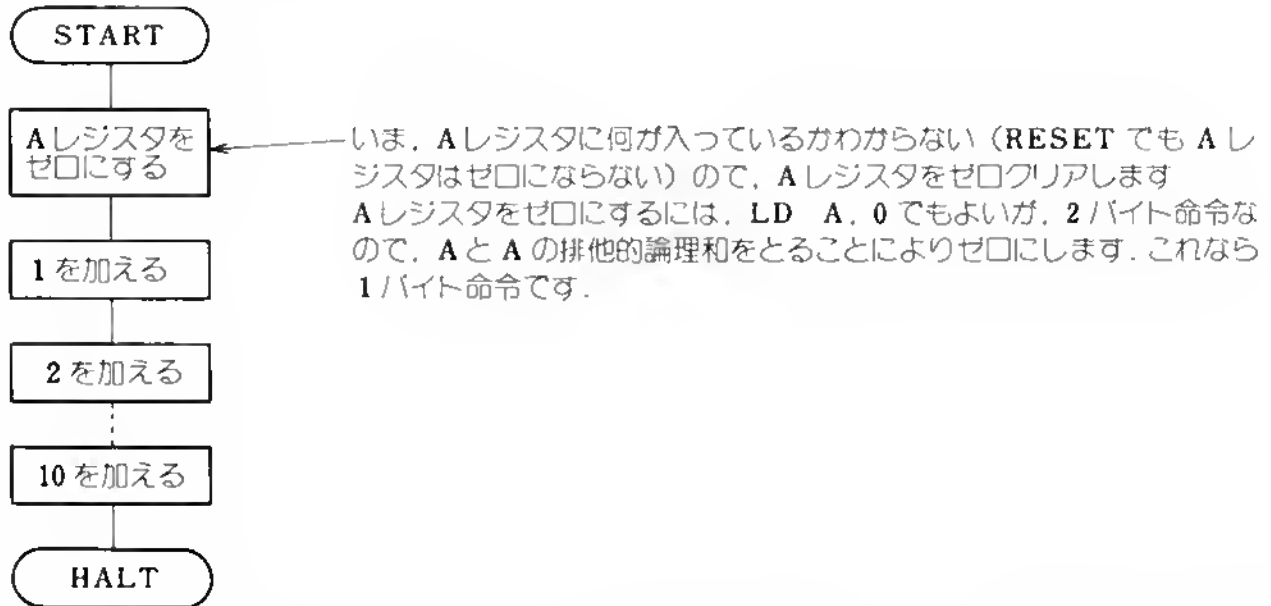
各社半導体規格表

## 例題

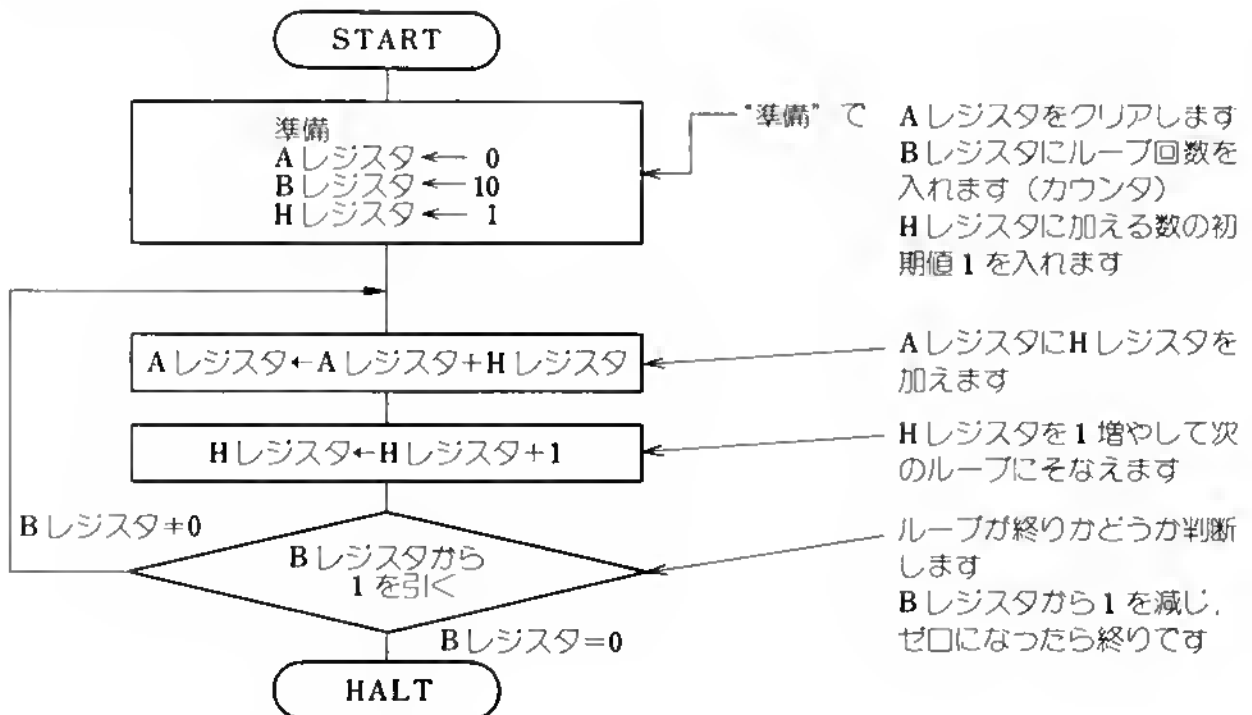
- プログラム 1 (ループ)
- プログラム 2 (判 断)
- プログラム 3 (メモリクリアサブルーチン)
- プログラム 4 (変換-テーブルサーチ)
- プログラム 5 (スイッチの表示)
- プログラム 6 (スイッチの表示-割り込み)

# プログラム 1 (ループ)

1 から 10 までの整数を加えるプログラムを考えます。答は A レジスタに入っているといえはよいとします。〈SAMPLE 1〉



これが 1~100 までだったら上のやり方では大変です。そこで「くり返し」を使います。〈SAMPLE 2〉



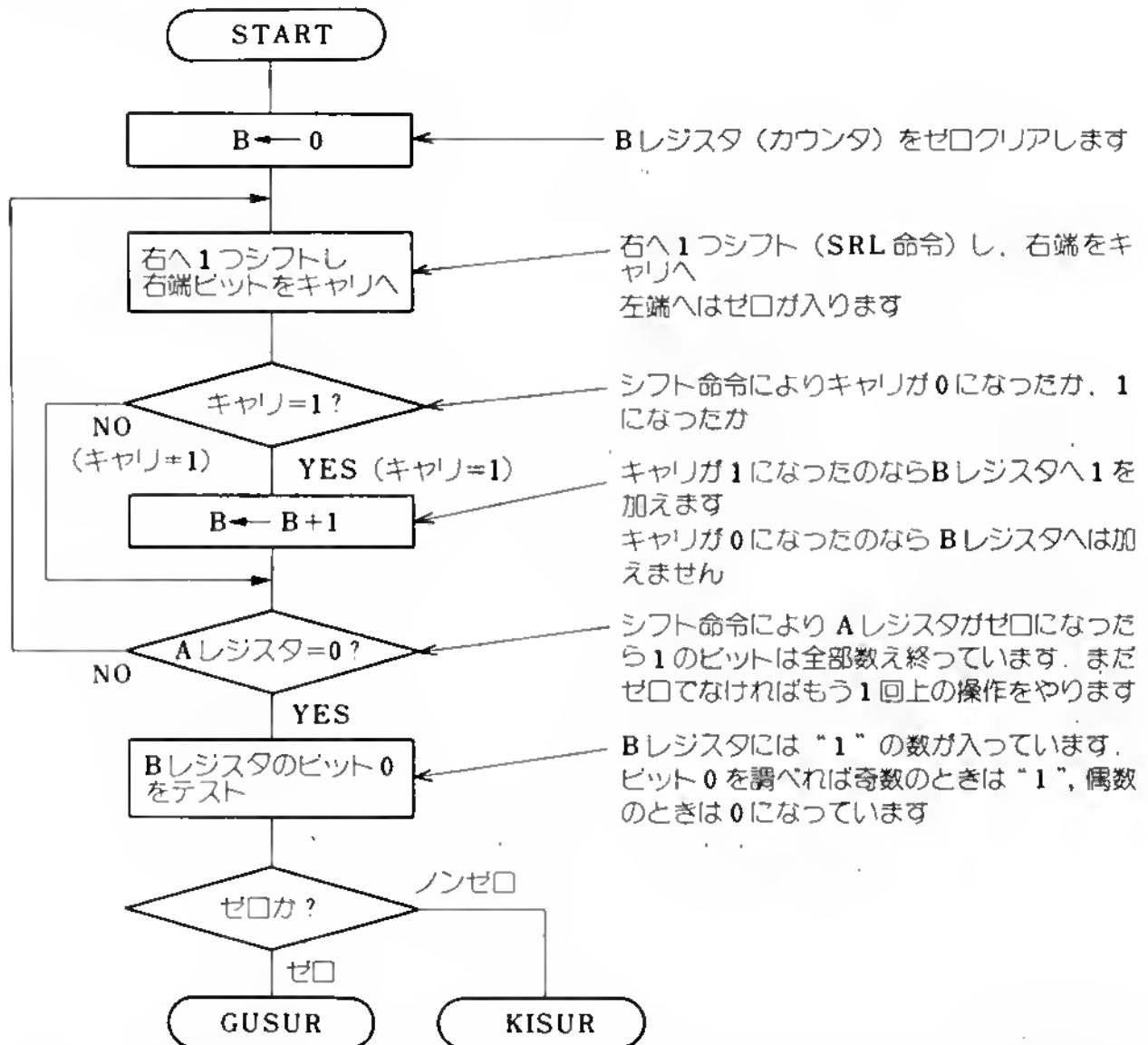
この方法なら、B レジスタに入れる数を変えればいくつでも加えられます。ただし、このままでは A レジスタがオーバーフローすることがありますので工夫がいらします。

[注] サンプルプログラムを全部一連にしてアセンブルしたものです、各サンプルプログラム間に関連はありません、実行するときにはモニタプログラムを持ったワンボードコンピュータ、たとえばシャープの SM-B-80TE を使って下さい。

行番号	ソースプログラム (キーを打ってインプットする)	ラベル	オペコード	オペランド	
	アドレス	マシン語			
1		プログラムを見やすくするためのコメント行			
2					
3					
4					
5			ORG	0000H	← このプログラムを 0000H 番地から配置することをアセンブラに知らせるアセンブラ命令
6	0000	AF	XOR	A	← A をゼロにする
7	0001	C601	ADD	A, 1	A ← A + 1 A は 1 になる
8	0003	C602	ADD	A, 2	A ← A + 2 A は 3 になる
9	0005	C603	ADD	A, 3	A ← A + 3 A は 6 になる
10	0007	C604	ADD	A, 4	
11	0009	C605	ADD	A, 5	
12	000B	C606	ADD	A, 6	
13	000D	C607	ADD	A, 7	
14	000F	C608	ADD	A, 8	
15	0011	C609	ADD	A, 9	
16	0013	C60A	ADD	A, 10	← ここで 1~10 までの和が A レジスタに入っている
17	0015	76	HALT	,	← CPU を止める
18					
19					
20					
21					
22	0016	AF	XOR	A	
23	0017	060A	LD	B, 10	カウンタの初期設定
24	0019	2601	LD	H, 1	ポインタの初期設定
25	0018	84	LOOP1	ADD	A, H A ← A + H
26	001C	24	INC	H	H ← H + 1
27	001D	10FC	DJNZ	LOOP1	← B ← B - 1 { B がノンゼロなら LOOP1 番地へ、ゼロなら次の番地へ
28	001F	76	HALT		

## プログラム 2 (判断)

A レジスタに入っているデータのビットパターンのうち“1”の数がいくつあるか数えます。結果が偶数なら GUSUR 番地へ、奇数なら KISUR 番地へジャンプするようにします。〈SAMPLE 3〉



もっとうまい方法があります。A と A の AND をとれば A は変わらずに、そのときの A の 1 の数に応じてパリティ・フラグがセットされます。

AND A ← A と A のアンドにより F レジスタをセットします

JP PE, GUSUR ← PE, はパリティ・イーブンすなわち A レジスタの「1」の数が偶数であればジャンプせよ」の条件付ジャンプ命令です

JP KISUR

これでも上と同じ働きをします。F レジスタだけ変わりますが、他のレジスタの内容はこのルーチンに入る前のまま変化しません。



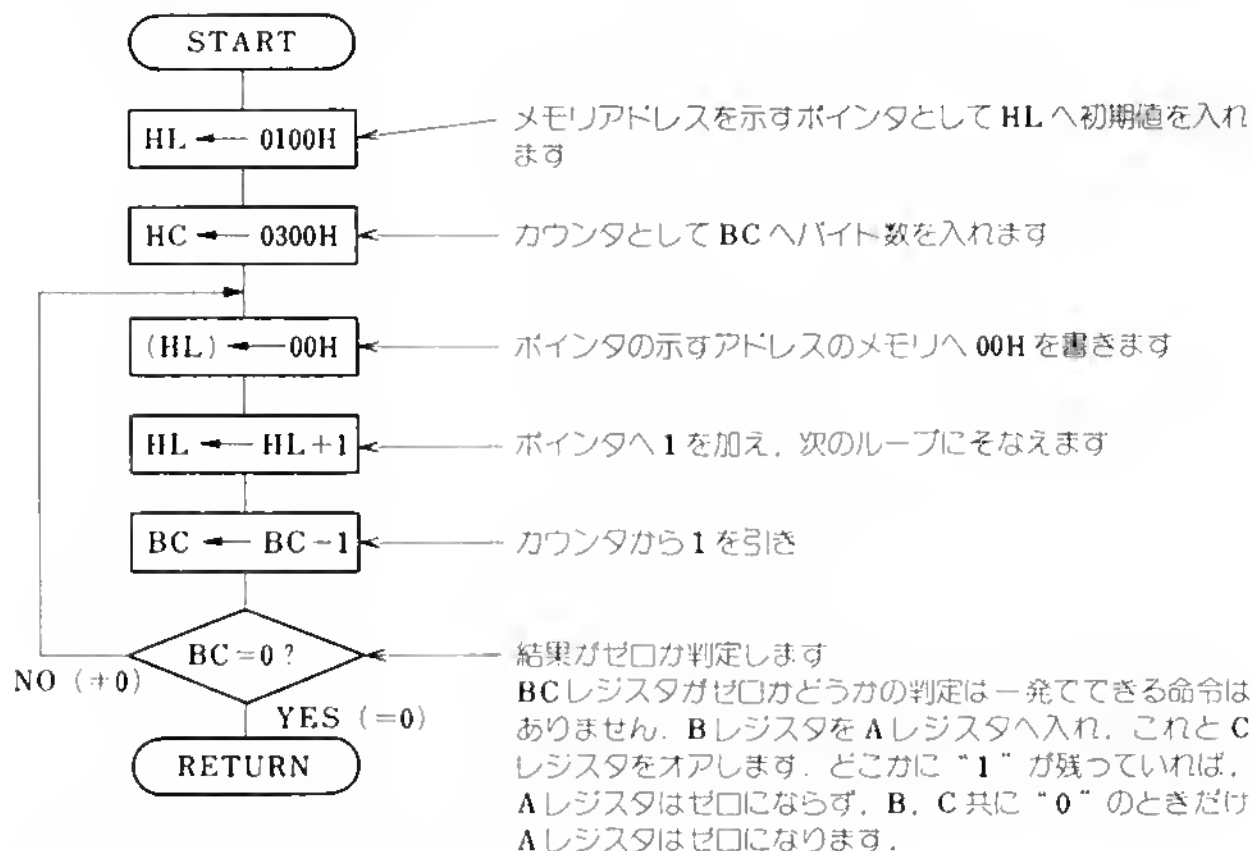
```

29          ; *****
30          ; SAMPLE 3
31          ;
32          ;
33          0200  GUSUR EQU 0200H ← { EQU は GUSUR というラベル名を 0200 H 番地へ
34          025F  KISUR EQU 025FH      { 結びつけるためのアセンブラ命令
35 0020 0600      LD B,0
36 0022 C83F      LOOP2 SRL A
37 0024 3001      JR NC,JP1
38 0026 04        INC B
39 0027 20F9      JP1 JR NZ,LOOP2
40 0029 C840      BIT 0,B
41 002B CA0002     JP Z,GUSUR
42 002E C35F02     JP KISUR

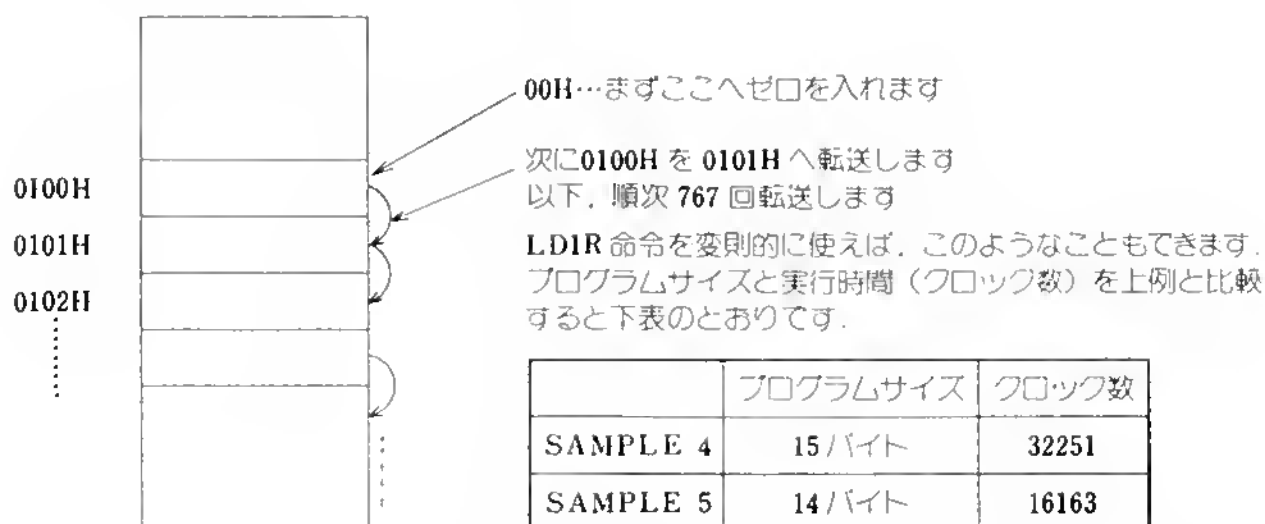
```

## プログラム 3 (メモリクリアサブルーチン)

0100H 番地から 03FFH 番地までの 300H バイト(=768 バイト)の RAM をゼロクリア (すべて 0 で埋めつくす) するサブルーチンを作ります。〈SAMPLE 4〉



別の方法もあります。〈SAMPLE 5〉



(プログラムサイズが小さく、実行が早いのがよいプログラムの条件です。ただしあまり名人芸的なプログラムを作るとあとで見たときに何をやっているのかわからなくなることがあります。メンテナンスやデバッグをしやすいのもよいプログラムの条件です)

```

43      ;*****
44      ;SAMPLE 4
45      ;
46      ;
47 0031 210001 MCLER LD HL,0100H
48 0034 010003      LD BC,0300H
49 0037 3600      ZEROM LD (HL),0
50 0039 23          INC HL
51 003A 08          DEC BC
52 003B 7B          LD A,B
53 003C B1          OR C
54 003D 20F8        JR NZ,ZEROM
55 003F C9          RET

```

} B, C 共にゼロであることを調べる

```

56      ;*****
57      ;SAMPLE5
58      ;
59      ;
60 0040 210001 MCLR1 LD HL,0100H
61 0043 110101      LD DE,0101H
62 0046 01FF02      LD BC,767
63 0049 3600          LD (HL),0
64 004B EDB0          LDIR
65 004D C9          RET

```

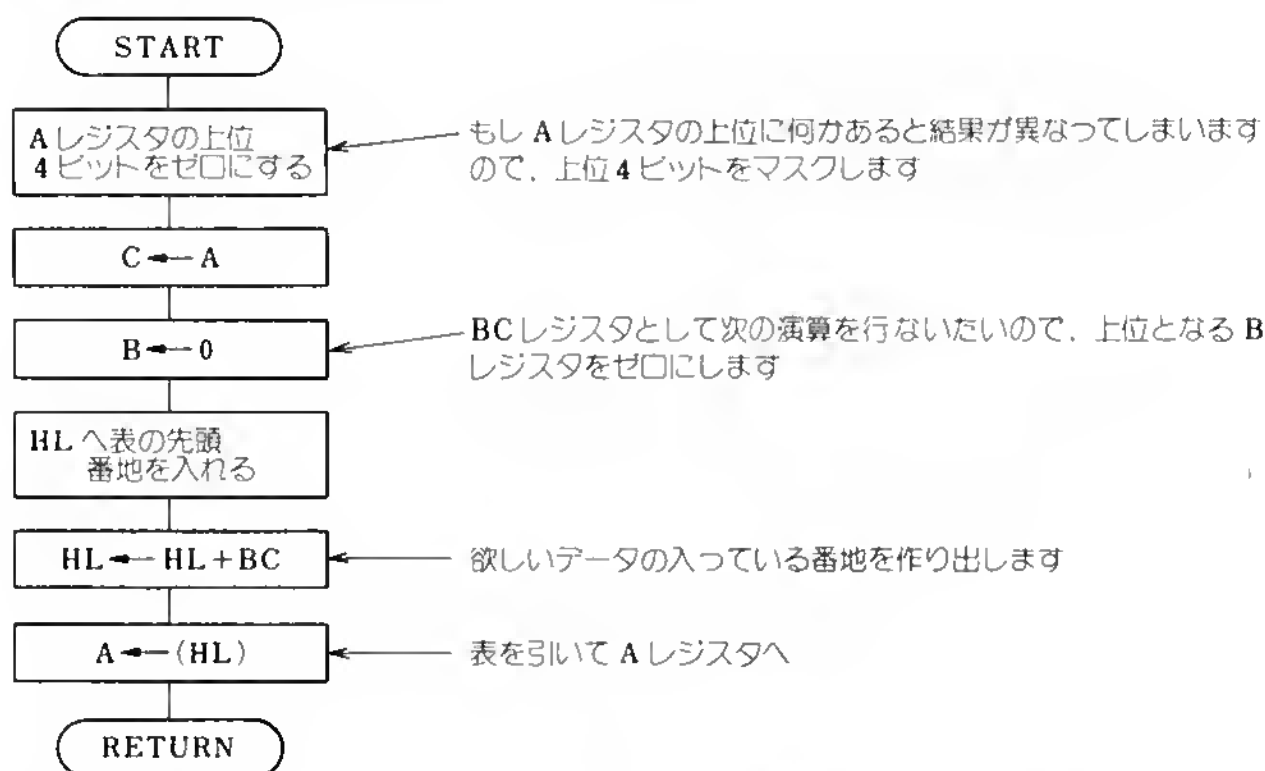
{ 10進数で定義してよい  
マシンの語は 02FF になっている

## プログラム 4 (変換-テーブルサーチ)

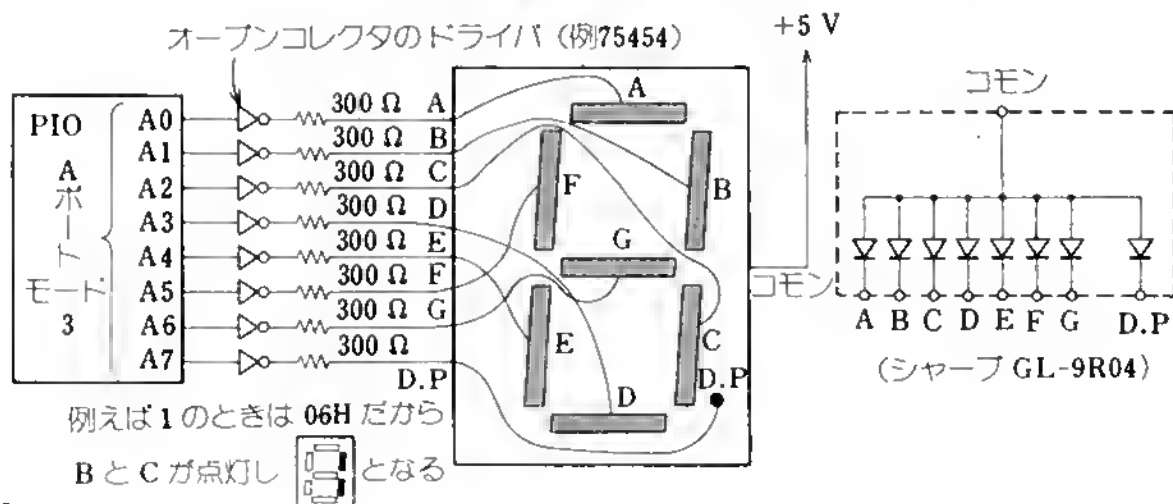
A レジスタの内容を表に従って変換します (下位 4 ビット対象)。〈SAMPLE 6〉

Aレジスタ	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
変換後	5C	06	5B	4F	66	6D	7D	27	7F	6F	77	7C	39	5E	79	71

変換前のデータが規則的で後は不規則です。したがって、前をメモリ・アドレスに対応させ、後をその内容とすれば、一発で表引きができます。もし相方共不規則なら 1 つ 1 つ一致するかどうか見て行くサーチの手法をとらなければなりません。



これは 4 ビットのデータを数字表示 LED に 16 進数表示するときを使うサブルーチンです。多桁表示したいときは、コモンを順次切り替えてダイナミック点灯させます。



```

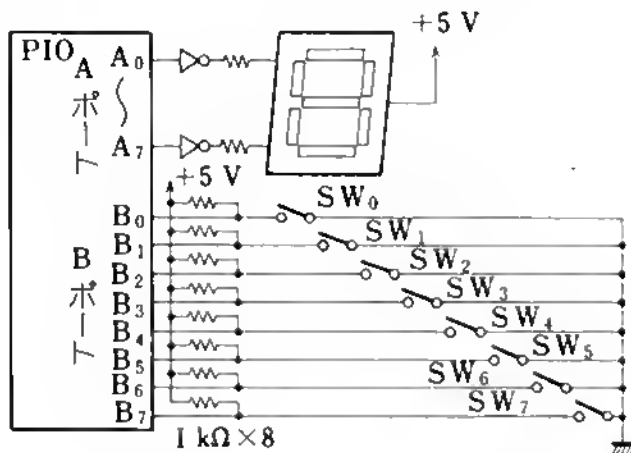
66          ; *****
67          ; SAMPLE6
68          ;
69          ;
70 004E E60F CONV    AND    0FH ← 上位4ビットをマスク
71 0050 4F      LD      C,A
72 0051 0600      LD      B,0
73 0053 215900    LD      HL, TABLE
74 0056 09      ADD     HL, BC ← HL に答の入っているアドレスを作成
75 0057 7E      LD      A, (HL)
76 0058 C9      RET
77 0059 5C      TABLE DEF B 5CH
78 005A 06      DEF B 06H
79 005B 5B      DEF B 5BH
80 005C 4F      DEF B 4FH
81 005D 66      DEF B 66H
82 005E 6D      DEF B 6DH
83 005F 7D      DEF B 7DH
84 0060 27      DEF B 27H
85 0061 7F      DEF B 7FH
86 0062 6F      DEF B 6FH
87 0063 77      DEF B 77H
88 0064 7C      DEF B 7CH
89 0065 39      DEF B 39H
90 0066 5E      DEF B 5EH
91 0067 79      DEF B 79H
92 0068 71      DEF B 71H

```

テーブル (表)  
 (DEFB はオペランドに書かれた1バイトの  
 値をそのままマシン語としてメモリへ定義す  
 るためのアセンブラ命令)

## プログラム 5 (スイッチの表示)

PIO に継がれた 8 個のスイッチのパターンを読み、前項のサブルーチンを使って LED に 16 進数表示させます。〈SAMPLE 7〉



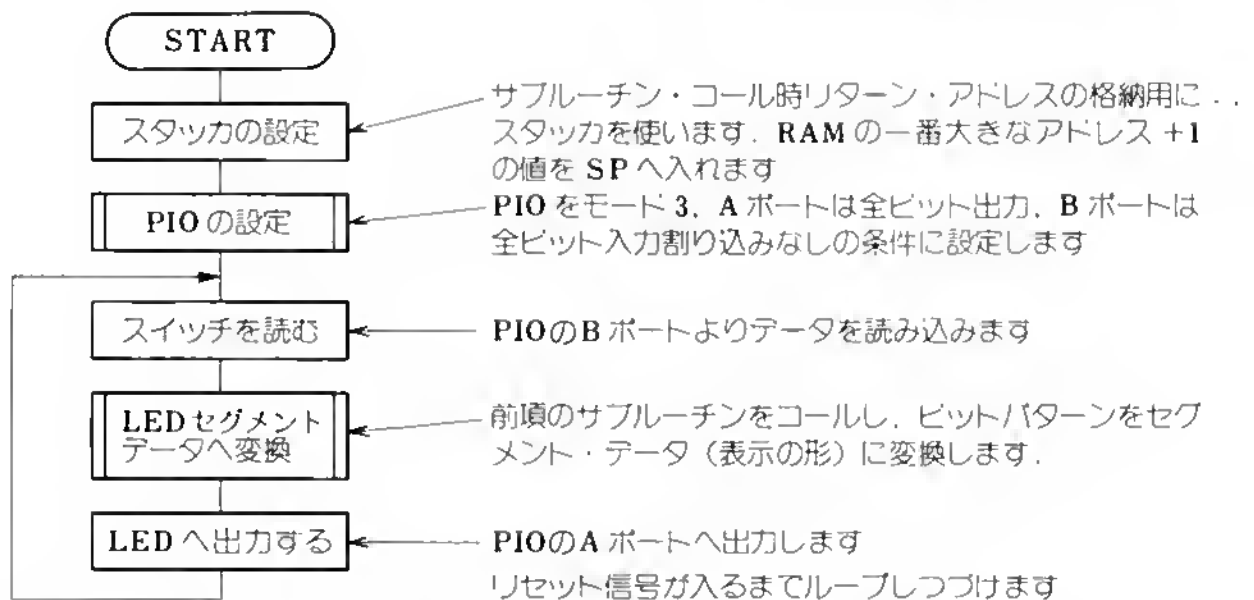
PIO のポートアドレスは下表とします

A ポート	データ	D0
A ポート	コントロール	D1
B ポート	データ	D2
B ポート	コントロール	D3

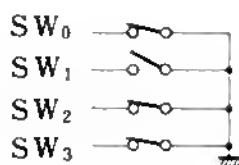
—○— = "1"

—●— = "0"

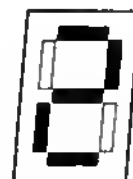
※SW<sub>4</sub>~SW<sub>7</sub>は、この例では使わない(無視される)



IN 命令を実行するたびにその時点の SW<sub>0</sub>~SW<sub>3</sub> の状態が 16 進表現で LED に表示されます。



なら 0010 だから



と表示されます

```

93      ; *****
94      ; SAMPLE7
95      ;
96      ;
97      0000      PIDAD1 EQU 000H
98      0001      PIOAC1 EQU 0D1H
99      0002      PID8D1 EQU 0D2H
100     0003      PI08C1 EQU 0D3H
101     0069 310000      LD SP,0000H
102     006C 218800      LD HL,INTPA
103     006F 0603      LD 8,3
104     0071 0ED1      LD C,PIDAC1
105     0073 ED83      OTIR
106     0075 218800      LD HL,INTPB
107     0078 0603      LD B,3
108     007A 0ED3      LD C,PI08C1
109     007C ED83      OTIR
110     007E D8D2      LOOP7 IN A,(PID8D1)
111     0080 CD4E00      CALL CDNU
112     0083 D3D0      OUT (PIDAD1),A
113     0085 C37E00      JP LDDP7
114     0088 CF      INTPA DEF8 0CFH
115     0089 00      DEF8 00H
116     008A 07      DEF8 07H
117     008B CF      INTPB DEF8 0CFH
118     008C FF      DEF8 0FFH
119     008D 07      DEF8 07H

```

ポートアドレスのラベル名定義  
 EQU は定義のためのアセンブラ命令  
 (マシン語には変換されない)

スタック設定、FFFF H 番地より前へ

PIO A ポート、制御語の書き込み

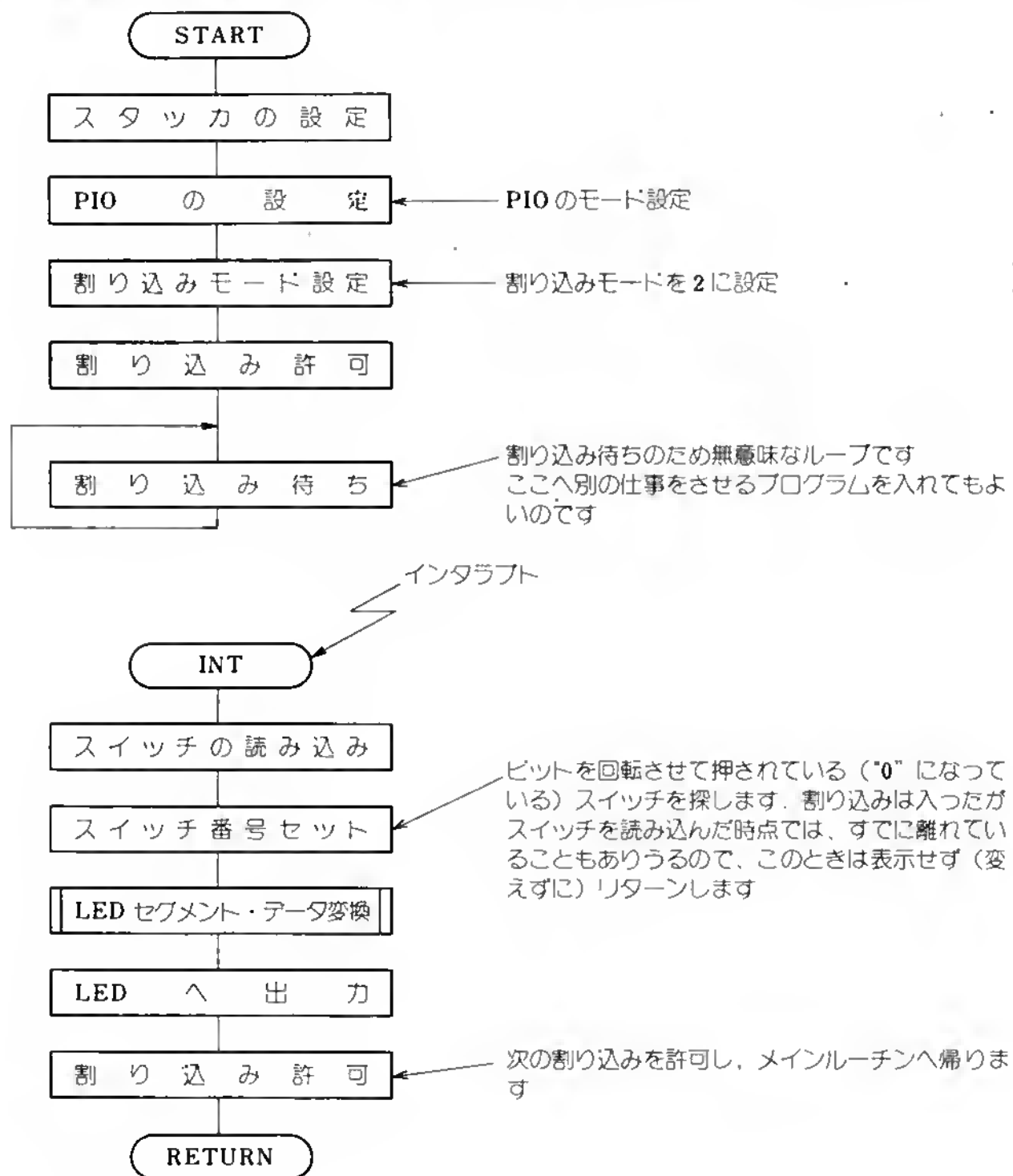
PIO B ポート、制御語の書き込み

スイッチ読み込み  
 データ変換をコール  
 出力

モード 3  
 全ビット出力  
 割り込みなし  
 モード 3  
 全ビット入力  
 割り込みなし

## プログラム 6 (スイッチの表示-割り込み)

前項のスイッチを押ボタンスイッチとして、押されたスイッチ番号 (SW<sub>0</sub> なら 0, SW<sub>3</sub> なら 3, SW<sub>7</sub> なら 7) を表示するように割り込みを使ってプログラムを作ります。ただし、同時に押したときは先に押したほうを優先します。全く同時なら番号の小さいほうを優先します。離しても次に押されるまで表示しつづけます。〈SAMPLE 8〉





```

120      ; *****
121      ; SAMPLEB
122      ;
123      ;
124      00D0      PIOAD      EQU      0D0H
125      00D1      PIOAC      EQU      0D1H
126      00D2      PIOBD      EQU      0D2H
127      00D3      PIOBC      EQU      0D3H
128      008E      310000      LD      SP, 0000H
129      0091      21AC00      LD      HL, INTLPA
130      0094      0603      LD      8, 3
131      0096      0ED1      LD      C, PIOAC
132      0098      ED83      OTIR
133      009A      21AF00      LD      HL, INTLPB
134      009D      0605      LD      8, 5
135      009F      0ED3      LD      C, PIOBC
136      00A1      ED83      OTIR
137      00A3      3E01      LD      A, 01H
138      00A5      ED47      LD      I, A
139      00A7      ED5E      IM      2
140      00A9      FB      EI
141      00AA      1BFE      WAIT      JR      WAIT
142      00AC      CF      INTLPA      DEFB      0CFH
143      00AD      00      DEFB      00H
144      00AE      07      DEFB      07H
145      00AF      00      INTLPB      DEFB      00H
146      0080      CF      DEFB      0CFH
147      00B1      FF      DEFB      0FFH
148      00B2      97      DEFB      97H
149      00B3      00      DEFB      00H
150      ORG      0100H
151      0100      0201      DEFW      INT
152      0102      D8D2      INT      IN      A, (PIOBD)
153      0104      0E00      LD      C, 0
154      0106      0608      LD      B, B
155      0108      1F      LOOPB      RRA
156      0109      3006      JR      NC, DISP
157      010B      0C      INC      C
158      010C      10FA      DJNZ      LOOPB
159      010E      FB      EI
160      010F      ED4D      RETI
161      0111      79      DISP      LD      A, C
162      0112      CD4E00      CALL      CONU
163      0115      D3D0      OUT      (PIOAD), A
164      0117      FB      EI
165      0118      ED4D      RETI
166      ; *****
167      END

```

ポートアドレスのラベル名定義

スタック設定、FFFF H 番地より前へ

PIO A ポート、制御語の書き込み

PIO B ポート、制御語の書き込み

I レジスタへ 01H

割り込みモード 2

割り込み許可

割り込み待ちのループ

モード 3

全ビット出力

割り込みなし

ベクトル 00H

モード 3

全ビット入力

・ 1

・ 2

・ 3

・ 4

[注] 下記参照

スウィッチの読み込み

右にループし右端ビットをキャリへ

キャリを調べ"0"のとき表示へ

8ビット共調べ、すべて"1"のときここでリターン

データ変換をコール

LEDへ出力

ソースプログラムの終りをアセンブラに知らせるアセンブラ命令

- ・ 1 割り込みあり、マスク制御語あり、オア、ローアクティブ
  - ・ 2 マスク制御語、全ビット割り込み発生
  - ・ 3 I レジスタとベクトルにより示される番地
  - ・ 4 割り込み処理ルーチンのエンタリーアドレス 0102 番地
- DEFW はオペランドの 2 バイトデータを定義するアセンブラ命令



# 付 録

## 付 1 Z-80 命 令 表

- $n$  は 8 ビット定数
- $lm$  は 16 ビット定数,  $l$  が上位 8 ビット,  $m$  が下位 8 ビット  
(ニーモニックではラベル名を書いてもよい)
- $d$  は  $-128 \sim +127$  までのディスプレイスメント
- $e$  は  $-128 \sim +127$  までのディスプレイスメント, 次の命令の先頭番地を 0 とする.  
(ニーモニックではラベル名か絶対番地を書く. ただしアセンブラにより異なる場合もある).
- $Cy$  は キャリ (C フラグ)
- 添字  $H$  は上位 8 ビット, 添字  $L$  は下位 8 ビットを意味する.
- ビット操作命令 SET RES BIT で, ビット番号は下記のとおりである.

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

左端

↑  $2^7$  ビット

右端

↑  $2^0$  ビット

- フラグ変化

- × 不定
- 1 1 になる
- 0 0 になる
- ・ 状態にしたがってセット, リセットされる
- 変化せず

ニーモニック	マシン語	フラグ変化						所要 クロック サイクル	動作	
		S	Z	H	P/V		N			C
					P	V				
ADC A, n	CE <u>n</u>							7	8ビット加算キャリ付 (アド・ウィズ・キャリ) A←A+ソース+Cy	
ADC A, A	8F							4		
ADC A, B	88									
ADC A, C	89									
ADC A, D	8A									
ADC A, E	8B	.	.	.	-	.	0	7		
ADC A, H	8C									
ADC A, L	8D									
ADC A, (HL)	8E									
ADC A, (IX+d)	DD 8E <u>d</u>							19		
ADC A, (IY+d)	FD 8E <u>d</u>							19		
ADC HL, BC	ED 4A							15	16ビット加算キャリ付 (アド・ウィズ・キャリ) HL←HL+ソース+Cy	
ADC HL, DE	ED 5A	.	.	x	-	.	0			
ADC HL, HL	ED 8A									
ADC HL, SP	ED 7A									
ADD A, n	C8 <u>n</u>							7	8ビット加算 (アド) A←A+ソース	
ADD A, A	87							4		
ADD A, B	80									
ADD A, C	81									
ADD A, D	82									
ADD A, E	83	.	.	.	-	.	0	7		
ADD A, H	84									
ADD A, L	85									
ADD A, (HL)	86									
ADD A, (IX+d)	DD 88 <u>d</u>							19		
ADD A, (IY+d)	FD 88 <u>d</u>							19		
ADD HL, BC	09							11	16ビット加算 (アド) HL←HL+ソース	
ADD HL, DE	19									
ADD HL, HL	29									
ADD HL, SP	39									
ADD IX, BC	DD 09							15	IX←IX+ソース	
ADD IX, DE	DD 19	-	-	x	-	-	0			
ADD IX, IX	DD 29									
ADD IX, SP	DD 39									
ADD IY, BC	FD 09							15	IY←IY+ソース	
ADD IY, DE	FD 19									
ADD IY, IY	FD 29									
ADD IY, SP	FD 39									
AND n	E8 <u>n</u>							7	論理積 (アンド) A←A∧ソース	
AND A	A7							4		
AND B	A0									
AND C	A1									
AND D	A2									
AND E	A3	.	.	1	.	-	0 0	7		
AND H	A4									
AND L	A5									
AND (HL)	A6									
AND (IX+d)	DD A6 <u>d</u>							19		
AND (IY+d)	FD A6 <u>d</u>							19		

a	b	答
0	0	0
0	1	0
1	0	0
1	1	1

ニーモニック	マシン語	フラグ変化							所要 クロック サイクル	動作
		S	Z	H	P/V	P/V	N	C		
BIT 0,A	CB 47								8	ビットテスト  ソースの第0ビットを調べ Zフラグを設定
BIT 0,B	CB 40									
BIT 0,C	CB 41									
BIT 0,D	CB 42									
BIT 0,E	CB 43									
BIT 0,H	CB 44	×		1	×		0	—		
BIT 0,L	CB 45									
BIT 0,(HL)	CB 46								12	
BIT 0,(IX+d)	DD CB_d_46								20	
BIT 0,(IY+d)	FD CB_d_46								20	
BIT 1,A	CB 4F								8	ビットテスト  ソースの第1ビットを調べ Zフラグを設定
BIT 1,B	CB 48									
BIT 1,C	CB 49									
BIT 1,D	CB 4A									
BIT 1,E	CB 4B									
BIT 1,H	CB 4C	×		1	×		0	—		
BIT 1,L	CB 4D									
BIT 1,(HL)	CB 4E								12	
BIT 1,(IX+d)	DD CB_d_4E								20	
BIT 1,(IY+d)	FD CB_d_4E								20	
BIT 2,A	CB 57								8	ビットテスト  ソースの第2ビットを調べ Zフラグを設定
BIT 2,B	CB 50									
BIT 2,C	CB 51									
BIT 2,D	CB 52									
BIT 2,E	CB 53									
BIT 2,H	CB 54	×		1	×		0	—		
BIT 2,L	CB 55									
BIT 2,(HL)	CB 56								12	
BIT 2,(IX+d)	DD CB_d_56								20	
BIT 2,(IY+d)	FD CB_d_56								20	
BIT 3,A	CB 5F								8	ビットテスト  ソースの第3ビットを調べ Zフラグを設定
BIT 3,B	CB 58									
BIT 3,C	CB 59									
BIT 3,D	CB 5A									
BIT 3,E	CB 5B									
BIT 3,H	CB 5C	×		1	×		0	—		
BIT 3,L	CB 5D									
BIT 3,(HL)	CB 5E								12	
BIT 3,(IX+d)	DD CB_d_5E								20	
BIT 3,(IY+d)	FD CB_d_5E								20	
BIT 4,A	CB 67								8	ビットテスト  ソースの第4ビットを調べ Zフラグを設定
BIT 4,B	CB 60									
BIT 4,C	CB 61									
BIT 4,D	CB 62									
BIT 4,E	CB 63									
BIT 4,H	CB 64	×		1	×		0	—		
BIT 4,L	CB 65									
BIT 4,(HL)	CB 66								12	
BIT 4,(IX+d)	DD CB_d_66								20	
BIT 4,(IY+d)	FD CB_d_66								20	

ニーモニック	マシン語	フラグ変化						所要 クロック サイクル	動作
		S	Z	H	P/V P/V	N	C		
BIT 5, A BIT 5, B BIT 5, C BIT 5, D BIT 5, E BIT 5, H BIT 5, L BIT 5, (HL) BIT 5, (IX+d) BIT 5, (IY+d)	CB 6F CB 68 CB 69 CB 6A CB 6B CB 6C CB 6D CB 6E DD CB <u>d</u> 6E FD CB <u>d</u> 6E								ビットテスト  ソースの第5ビットを調べ Zフラグを設定
BIT 6, A BIT 6, B BIT 6, C BIT 6, D BIT 6, E BIT 6, H BIT 6, L BIT 6, (HL) BIT 6, (IX+d) BIT 6, (IY+d)	CB 77 CB 70 CB 71 CB 72 CB 73 CB 74 CB 75 CB 76 DD CB <u>d</u> 76 FD CB <u>d</u> 76								ビットテスト  ソースの第6ビットを調べ Zフラグを設定
BIT 7, A BIT 7, B BIT 7, C BIT 7, D BIT 7, E BIT 7, H BIT 7, L BIT 7, (HL) BIT 7, (IX+d) BIT 7, (IY+d)	CB 7F CB 78 CB 79 CB 7A CB 7B CB 7C CB 7D CB 7E DD CB <u>d</u> 7E FD CB <u>d</u> 7E								ビットテスト  ソースの第7ビットを調べ Zフラグを設定
CALL NZ, <u>lm</u> CALL Z, <u>lm</u> CALL NC, <u>lm</u> CALL C, <u>lm</u> CALL PQ, <u>lm</u> CALL PE, <u>lm</u> CALL P, <u>lm</u> CALL M, <u>lm</u>	C4 <u>m</u> <u>l</u> CC <u>m</u> <u>l</u> D4 <u>m</u> <u>l</u> DC <u>m</u> <u>l</u> E4 <u>m</u> <u>l</u> EC <u>m</u> <u>l</u> F4 <u>m</u> <u>l</u> FC <u>m</u> <u>l</u>							成立時 17 不成立 10	サブルーチン・コール(条件付) ・条件が成立すれば戻り番地 [PC] をスタ ックへ PUSH し <u>lm</u> へジャンプ [PC ← <u>lm</u> ] ・成立しなければ本命令は無視する
CALL <u>lm</u>	CD <u>m</u> <u>l</u>							17	サブルーチン・コール(無条件) PC をスタックへ PUSH し PC ← <u>lm</u>
CCF	3F							4	Cy を反転 [Cy ← Cy]
CP n CP A CP B CP C CP D CP E CP H CP L CP (HL) CP (IX+d) CP (IY+d)	FE <u>n</u> BF B8 B9 BA BB BC BD BE DD BE <u>d</u> FD BE <u>d</u>							7 4 7 19 19	比較(コンペア) A - ソースの演算をする A の内容は変わらずフラグだけが変化する

ニーモニック	マシン語	フラグ変化							所 要 クロック サイクル	動作	
		S	Z	H	P/V		N	C			
					P	V					
CPD	ED A9	×	×	×	—	—	1	—	16	比較(コンペア・ディクリメント) A←(HL) のフラグ変化のみ HL←HL-1 BC←BC-1	
CPDR	ED B9	×	×	×	—	—	1	—	1バイトにつき 21 最終のみ 16	比較(コンペア・ディクリメント・リピート) CPDをA←(HL) [Zフラグ=1] または BC=0 [Vフラグ=0] までくり返す	
CP1	ED A1	×	×	×	—	—	1	—	16	比較(コンペア・インクリメント) A←(HL) のフラグ変化のみ HL←HL+1 BC←BC+1	
CPIR	ED B1	×	×	×	—	—	1	—	1バイトにつき 21 最終のみ 16	比較(コンペア・インクリメント・リピート) CPIをA←(HL) [Zフラグ=1] または BC=0 [Vフラグ=0] までくり返す	
CPL	2F	—	—	1	—	—	1	—	4	コンプリメント Aレジスタに対しビット反転 0→1 1→0	
DAA	27	—	—	—	—	—	—	—	4	デシマル・アジャスト・アキュムレータ Aレジスタに対し10進補正	
DEC A	3D	—	—	—	—	—	—	—	4	8ビットデクリメント  ソース←ソース-1	
DEC B	05	—	—	—	—	—	—	—			
DEC C	0D	—	—	—	—	—	—	—			
DEC D	15	—	—	—	—	—	—	—			
DEC E	1D	—	—	—	—	—	—	—			
DEC H	25	—	—	—	—	—	1	—			
DEC L	2D	—	—	—	—	—	—	—			
DEC (HL)	35	—	—	—	—	—	—	—			
DEC (IX+d)	DD 35 d	—	—	—	—	—	—	—	11	16ビットデクリメント  ソース←ソース-1	
DEC (IY+d)	FD 35 d	—	—	—	—	—	—	—	23		
DEC BC	0B	—	—	—	—	—	—	—	6		
DEC DE	1B	—	—	—	—	—	—	—			
DEC HL	2B	—	—	—	—	—	—	—			
DEC SP	3B	—	—	—	—	—	—	—			
DEC IX	DD 2B	—	—	—	—	—	—	—			10
DEC IY	FD 2B	—	—	—	—	—	—	—			10
DI	F3	—	—	—	—	—	—	—	4	割り込み禁止(ディセーブル・インタラプト)	
DJNZ e	10 e	—	—	—	—	—	—	—	B=0 8 B≠0 13	(デクリメント・ジャンプノンゼロ) B←B-1 B≠0ならeバイトだけジャンプ [PC←PC+e] B=0ならジャンプせず [e=0と同じ]	
EI	FB	—	—	—	—	—	—	—	4	割り込み許可(イネーブル・インタラプト)	
EX (SP), HL	E3	—	—	—	—	—	—	—	19	交換(エクスチェンジ) ソースとデスティネーションの内容を交換する	
EX (SP), IX	DD E3	—	—	—	—	—	—	—	23		
EX (SP), IY	FD E3	—	—	—	—	—	—	—	23		
EX AF, AF'	08	—	—	—	—	—	—	—	4		
EX DE, HL	EB	—	—	—	—	—	—	—	4		
EXX	D9	—	—	—	—	—	—	—	4	BC DE HLの内容と BC' DE' HL'の内容を交換する	

ニーモニク	マシン語	フラグ変化						所要 クロック サイクル	動作
		S	Z	H	P/V P V	N	C		
HALT	76	-	-	-	-	-	-	4	命令実行の進行を止めリセットまたは割り込み待ちとなる (ホルト)
IM 0	ED 46	-	-	-	-	-	-	8	割り込みモードを0に設定する
IM 1	ED 56	-	-	-	-	-	-	8	割り込みモードを1に設定する
IM 2	ED 5E	-	-	-	-	-	-	8	割り込みモードを2に設定する
INC A	3C	-	-	-	-	-	-	4	8ビットインクリメント  ソース ← ソース + 1
INC B	04	-	-	-	-	-	-		
INC C	0C	-	-	-	-	-	-		
INC D	14	-	-	-	-	-	-		
INC E	1C	-	-	-	-	0	-	11	
INC H	24	-	-	-	-	-	-		
INC L	2C	-	-	-	-	-	-		
INC (HL)	34	-	-	-	-	-	-		
INC (IX+d)	DD 34 d	-	-	-	-	-	-	23	
INC (IY+d)	FD 34 d	-	-	-	-	-	-	23	
INC BC	03	-	-	-	-	-	-	6	16ビットインクリメント  ソース ← ソース + I
INC DE	13	-	-	-	-	-	-		
INC HL	23	-	-	-	-	-	-		
INC SP	33	-	-	-	-	-	-		
INC IX	DD 23	-	-	-	-	-	-	10	
INC IY	FD 23	-	-	-	-	-	-	10	
IN A, (C)	ED 78	-	-	-	-	-	-	12	入力 Cレジスタの内容番地のポートからディスティネーションのレジスタへ入力 [ディスティネーション ← (C)]
IN B, (C)	ED 40	-	-	-	-	-	-		
IN C, (C)	ED 48	-	-	-	-	-	-		
IN D, (C)	ED 50	-	-	-	-	-	-		
IN E, (C)	ED 58	-	-	-	-	-	-		
IN H, (C)	ED 60	-	-	-	-	-	-		
IN L, (C)	ED 68	-	-	-	-	-	-	11	入力 n番地のポートからAレジスタへ
IN A, (n)	DB n	-	-	-	-	-	-		
IND	ED AA	x	-	x	x	-	x	16	イン・ディクリメント (HL) ← (C) HL ← HL - 1 B ← B - 1
INDR	ED BA	x	1	x	x	-	x	1バイトにつき 21 最終のみ 16	イン・ディクリメント・リピート INDをB=0までくり返す
INI	ED A2	x	-	x	x	-	x	16	イン・インクリメント (HL) ← (C) HL ← HL + 1 B ← B - 1
INIR	ED B2	x	1	x	x	-	x	1バイトにつき 21 最終のみ 16	イン・インクリメント・リピート INIをB=0までくり返す
JP (HL)	E9	-	-	-	-	-	-	4	ジャンプ (無条件)
JP (IX)	DD E9	-	-	-	-	-	-	8	各レジスタの内容番地へジャンプ [PC ← HL]
JP (IY)	FD E9	-	-	-	-	-	-	8	
JP Im	C3 m l	-	-	-	-	-	-	10	



ニーモニック	マシン語	フラグ変化							所 要 クロック サイクル	動 作
		S	Z	H	P/V		N	C		
					P	V				
JP NZ, <u>lm</u>	C2 <u>m</u> <u>l</u>								10	ジャンプ (条件付) ・条件が成立すれば <u>lm</u> 番地へジャンプ (PC ← <u>lm</u> ) ・不成立なら本命令は無視する
JP Z, <u>lm</u>	CA <u>m</u> <u>l</u>									
JP NC, <u>lm</u>	D2 <u>m</u> <u>l</u>									
JP C, <u>lm</u>	DA <u>m</u> <u>l</u>									
JP PO, <u>lm</u>	E2 <u>m</u> <u>l</u>									
JP PE, <u>lm</u>	EA <u>m</u> <u>l</u>									
JP P, <u>lm</u>	F2 <u>m</u> <u>l</u>									
JP M, <u>lm</u>	FA <u>m</u> <u>l</u>									
JR e	18 <u>e</u>								12	ジャンプ・リラティブ (無条件) e バイト先へジャンプする [PC ← PC + e]
JR NZ, e	20 <u>e</u>								条件成立 12 条件不成立 7	ジャンプ・リラティブ (条件付) ・条件が成立すれば e バイト先へジャンプ [PC ← PC + e] ・不成立なら本命令は無視する
JR Z, e	28 <u>e</u>									
JR NC, e	30 <u>e</u>									
JR C, e	38 <u>e</u>									
LD A, n	3E <u>n</u>								7	8ビット転送 (ロード)  A ← ソース
LD A, A	7F								4	
LD A, B	78									
LD A, C	79									
LD A, D	7A									
LD A, E	7B								13	
LD A, H	7C									
LD A, L	7D									
LD A, ( <u>lm</u> )	3A <u>m</u> <u>l</u>								7	
LD A, (BC)	0A								7	
LD A, (DE)	1A								7	
LD A, (HL)	7E								7	
LD A, (IX+d)	DD 7E <u>d</u>								19	
LD A, (IY+d)	FD 7E <u>d</u>								19	
LD A, I	ED 57								9	8ビット転送 A ← I A ← R IFF: 0 のとき割り込み禁止 (DI) 1 のとき割り込み可 (EI) になっている LD A, I LD A, R ではこの値が P/V にコピーされる
LD A, R	ED 5F								9	
LD B, n	06 <u>n</u>								7	8ビット転送 (ロード)  B ← ソース
LD B, A	47								4	
LD B, B	40									
LD B, C	41									
LD B, D	42									
LD B, E	43								7	
LD B, H	44									
LD B, L	45									
LD B, (HL)	46								19	
LD B, (IX+d)	DD 46 <u>d</u>								19	
LD B, (IY+d)	FD 46 <u>d</u>								19	

ニーモニック		マシン語	フラグ変化						所 要 クロック サイクル	動 作	
			S	Z	H	P/V		N			C
						P	V				
LD	C, n	0E <u>n</u>							7	8ビット転送 (ロード)  C ← ソース	
LD	C, A	4F							4		
LD	C, B	48									
LD	C, C	49									
LD	C, D	4A									
LD	C, E	4B	-	-	-	-	-	-	7		
LD	C, H	4C									
LD	C, L	4D									
LD	C, (HL)	4E									
LD	C, (IX+d)	DD 4E <u>d</u>							19		
LD	C, (IY+d)	FD 4E <u>d</u>							19		
LD	D, n	16 <u>n</u>							7	8ビット転送 (ロード)  D ← ソース	
LD	D, A	57							4		
LD	D, B	50									
LD	D, C	51									
LD	D, D	52									
LD	D, E	53	-	-	-	-	-	-	7		
LD	D, H	54									
LD	D, L	55									
LD	D, (HL)	56									
LD	D, (IX+d)	DD 56 <u>d</u>							19		
LD	D, (IY+d)	FD 56 <u>d</u>							19		
LD	E, n	1E <u>n</u>							7	8ビット転送 (ロード)  E ← ソース	
LD	E, A	5F							4		
LD	E, B	58									
LD	E, C	59									
LD	E, D	5A									
LD	E, E	5B	-	-	-	-	-	-	7		
LD	E, H	5C									
LD	E, L	5D									
LD	E, (HL)	5E									
LD	E, (IX+d)	DD 5E <u>d</u>							19		
LD	E, (IY+d)	FD 5E <u>d</u>							19		
LD	H, n	26 <u>n</u>							7	8ビット転送 (ロード)  H ← ソース	
LD	H, A	67							4		
LD	H, B	60									
LD	H, C	61									
LD	H, D	62									
LD	H, E	63	-	-	-	-	-	-	7		
LD	H, H	64									
LD	H, L	65									
LD	H, (HL)	66									
LD	H, (IX+d)	DD 66 <u>d</u>							19		
LD	H, (IY+d)	FD 66 <u>d</u>							19		

ニーモニック		マシン語	フラグ変化						所 要 クロック サイクル	動 作	
			S	Z	H	P/V		N			C
						P	V				
LD	L, n	2E <u>n</u>							7	8ビット転送 (ロード)  L ← ソース	
LD	L, A	6F							4		
LD	L, B	68									
LD	L, C	69									
LD	L, D	6A									
LD	L, E	6B	-	-	-	-	-	-			
LD	L, H	6C							7		
LD	L, L	6D									
LD	L, (HL)	6E									
LD	L, (IX+d)	DD 6E <u>d</u>							19		
LD	L, (IY+d)	FD 6E <u>d</u>							19		
LD	I, A	ED 47	-	-	-	-	-	-	9	8ビット転送 I ← A	
LD	R, A	ED 4F	-	-	-	-	-	-	9		R ← A
LD	(Im), A	32 <u>m</u> <u>I</u>							13	8ビット転送 (Im) ← A	
LD	(BC), A	02	-	-	-	-	-	-	7		(BC) ← A
LD	(DE), A	12	-	-	-	-	-	-	7		(DE) ← A
LD	(HL), n	36 <u>n</u>							10	8ビット転送 (ロード)  (HL) ← ソース	
LD	(HL), A	77							7		
LD	(HL), B	70									
LD	(HL), C	71									
LD	(HL), D	72	-	-	-	-	-	-			
LD	(HL), E	73									
LD	(HL), H	74									
LD	(HL), L	75									
LD	(IX+d), n	DD 36 <u>d</u> <u>n</u>							19	8ビット転送 (ロード)  (IX+d) ← ソース	
LD	(IX+d), A	DD 77 <u>d</u>									
LD	(IX+d), B	DD 70 <u>d</u>									
LD	(IX+d), C	DD 71 <u>d</u>									
LD	(IX+d), D	DD 72 <u>d</u>	-	-	-	-	-	-			
LD	(IX+d), E	DD 73 <u>d</u>									
LD	(IX+d), H	DD 74 <u>d</u>									
LD	(IX+d), L	DD 75 <u>d</u>									
LD	(IY+d), n	FD 36 <u>d</u> <u>n</u>							19	8ビット転送 (ロード)  (IY+d) ← ソース	
LD	(IY+d), A	FD 77 <u>d</u>									
LD	(IY+d), B	FD 70 <u>d</u>									
LD	(IY+d), C	FD 71 <u>d</u>									
LD	(IY+d), D	FD 72 <u>d</u>	-	-	-	-	-	-			
LD	(IY+d), E	FD 73 <u>d</u>									
LD	(IY+d), H	FD 74 <u>d</u>									
LD	(IY+d), L	FD 75 <u>d</u>									
LD	BC, Im	01 <u>m</u> <u>I</u>	-	-	-	-	-	-	10	16ビット転送 BC ← ソース	〔Im:定数 (Im):メモリの内容〕 メモリからレジスタの場合 たとえば HL, (Im)では L ← (Im) H ← (Im+1) となる
LD	BC, (Im)	ED 4B <u>m</u> <u>I</u>	-	-	-	-	-	-	20		
LD	DE, Im	11 <u>m</u> <u>I</u>	-	-	-	-	-	-	10	16ビット転送 DE ← ソース	
LD	DE, (Im)	ED 5B <u>m</u> <u>I</u>	-	-	-	-	-	-	20		
LD	HL, Im	21 <u>m</u> <u>I</u>	-	-	-	-	-	-	10	16ビット転送 HL ← ソース	
LD	HL, (Im)	2A <u>m</u> <u>I</u>	-	-	-	-	-	-	16		

{Im: 定数  
 (Im): メモリの内容}  
 メモリからレジスタの場合  
 たとえば  
 HL, (Im) では  
 L ← (Im)  
 H ← (Im+1)  
 となる

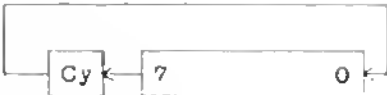


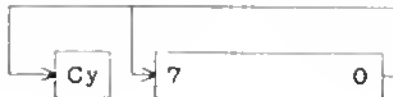
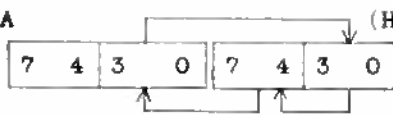
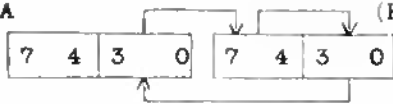
ニーモニック	マシン語	フラグ変化							所要 クロック サイクル	動作
		S	Z	H	P/V		N	C		
LD SP, <i>lm</i>	31 <u>m</u> <u>l</u>								10	16ビット転送
LD SP, ( <i>lm</i> )	ED 7B <u>m</u> <u>l</u>								20	
LD SP, HL	F9	-	-	-	-	-	-	-	6	SP ← ソース
LD SP, IX	DD F9								10	
LD SP, IY	FD F9								10	
LD IX, <i>lm</i>	DD 21 <u>m</u> <u>l</u>	-	-	-	-	-	-	-	14	16ビット転送
LD IX, ( <i>lm</i> )	DD 2A <u>m</u> <u>l</u>								20	IX ← ソース
LD IY, <i>lm</i>	FD 21 <u>m</u> <u>l</u>	-	-	-	-	-	-	-	14	16ビット転送
LD IY, ( <i>lm</i> )	FD 2A <u>m</u> <u>l</u>								20	IY ← ソース
LD ( <i>lm</i> ), BC	ED 43 <u>m</u> <u>l</u>								20	16ビット転送 (ロード)
LD ( <i>lm</i> ), DE	ED 53 <u>m</u> <u>l</u>								20	
LD ( <i>lm</i> ), HL	22 <u>m</u> <u>l</u>	-	-	-	-	-	-	-	16	( <i>lm</i> ) ← ソース L
LD ( <i>lm</i> ), SP	ED 73 <u>m</u> <u>l</u>								20	( <i>lm</i> + 1) ← ソース H
LD ( <i>lm</i> ), IX	DD 22 <u>m</u> <u>l</u>								20	
LD ( <i>lm</i> ), IY	FD 22 <u>m</u> <u>l</u>								20	
LDD	ED A8	×	×	0	-	0	-	-	16	ブロック転送 (ロード・ディクリメント) (DE) ← (HL) DE ← DE - 1 HL ← HL - 1 BC ← BC - 1
LDDR	ED B8	×	×	0	-	0	0	-	1バイトにつき 21 最終のみ 16	ブロック転送 (ロード・ディクリメント・リピート) LDDをBC = 0までくり返す
LDI	ED A0	×	×	0	-	0	-	-	16	ブロック転送 (ロード・インクリメント) (DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC - 1
LDIR	ED B0	×	×	0	-	0	0	-	1バイトにつき 21 最終のみ 16	ブロック転送 (ロード・インクリメント・リピート) LDIをBC = 0までくり返す
NEO	ED 44	.	.	.	-	1	.	.	8	ニゲイト Zの補数をとる A ← 0 - A
NOP	00	-	-	-	-	-	-	-	4	何もしないで次へ (ノーオペレーション)
OR <i>n</i>	F6 <u>n</u>								7	論理和 (オア)
OR A	B7									
OR B	B0									
OR C	B1									
OR D	B2									
OR E	B3	.	.	0	.	-	0	0	4	
OR H	B4									
OR L	B5									
OR (HL)	B6								7	
OR (IX+d)	DD B6 <u>d</u>								19	
OR (IY+d)	FD B6 <u>d</u>								19	

a	b	答
0	0	0
0	1	1
1	0	1
1	1	1

ニ ー モ ニ ッ ク	マ シ ン 語	フ ラ グ 変 化						所 要 クロック サイクル	動 作
		S	Z	H	P/V P V	N	C		
OUT (C), A	ED 79							12	出力  各レジスタの内容をCレジスタの内容番地のポートへ出力 [(C)←ソース]
OUT (C), B	ED 41								
OUT (C), C	ED 49								
OUT (C), D	ED 51	-	-	-	-	-	-		
OUT (C), E	ED 59								
OUT (C), H	ED 61								
OUT (C), L	ED 69								
OUT (n), A	D3 <u>n</u>	-	-	-	-	-	-	11	出力 Aレジスタの内容をn番地のポートへ
OUTD	ED AB	×	×	×	×	-	×	16	アウト・ディクリメント (C)←(HL) HL←HL-1 B←B-1
OTDR	ED BB	×	1	×	×	-	×	1バイト につき 21 最終のみ 16	アウト・ディクリメント・リピート  OUTDをB=0までくり返す
OUTI	ED A3	×	×	×	×	-	×	16	アウト・インクリメント (C)←(HL) HL←HL+1 B←B-1
OTIR	ED B3	×	1	×	×	-	×	1バイト につき 21 最終のみ 16	アウト・インクリメント・リピート  OUTIをB=0までくり返す
POP AF	F1							10    14 14	16ビット転送(ポップ) スタックからレジスタへ転送 ディスティネーション <sub>L</sub> ←(SP) ディスティネーション <sub>H</sub> ←(SP+1) SP←SP+2
POP BC	C1								
POP DE	D1	-	-	-	-	-	-		
POP HL	E1								
POP IX	DD E1								
POP IY	FD E1								
PUSH AF	F5							11    15 15	16ビット転送(プッシュ) レジスタからスタックへ転送 (SP-1)←ソース <sub>H</sub> (SP-2)←ソース <sub>L</sub> SP←SP-2
PUSH BC	C5								
PUSH DE	D5	-	-	-	-	-	-		
PUSH HL	E5								
PUSH IX	DD E5								
PUSH IY	FD E5								
RES 0, A	CB 87							8       15 23 23	ビットリセット   ソースの第0ビット←0
RES 0, B	CB 80								
RES 0, C	CB 81								
RES 0, D	CB 82								
RES 0, E	CB 83								
RES 0, H	CB 84	-	-	-	-	-	-		
RES 0, L	CB 85								
RES 0, (HL)	CB 86								
RES 0, (IX+d)	DD CB <u>d</u> 86								
RES 0, (IY+d)	FD CB <u>d</u> 86								

ニ ー モ ニ ッ ク	マ シ ン 語	フ ラ グ 変 化						所 要 クロック サイクル	動 作
		S	Z	H	P/V P V	N	C		
RES 1.A	CB 8F							8	ビットリセット  ソースの第1ビット←0
RES 1.B	CB 88								
RES 1.C	CB 89								
RES 1.D	CB 8A								
RES 1.E	CB 8B								
RES 1.H	CB 8C	-	-	-	-	-	-		
RES 1.L	CB 8D								
RES 1.(HL)	CB 8E							15	
RES 1.(IX+d)	DD CB <u>d</u> 8E							23	
RES 1.(IY+d)	FD CB <u>d</u> 8E							23	
RES 2.A	CB 97							8	ビットリセット  ソースの第2ビット←0
RES 2.B	CB 90								
RES 2.C	CB 91								
RES 2.D	CB 92								
RES 2.E	CB 93								
RES 2.H	CB 94	-	-	-	-	-	-		
RES 2.L	CB 95								
RES 2.(HL)	CB 96							15	
RES 2.(IX-d)	DD CB <u>d</u> 96							23	
RES 2.(IY-d)	FD CB <u>d</u> 96							23	
RES 3.A	CB 9F							8	ビットリセット  ソースの第3ビット←0
RES 3.B	CB 98								
RES 3.C	CB 99								
RES 3.D	CB 9A								
RES 3.E	CB 9B								
RES 3.H	CB 9C	-	-	-	-	-	-		
RES 3.L	CB 9D								
RES 3.(HL)	CB 9E							15	
RES 3.(IX+d)	DD CB <u>d</u> 9E							23	
RES 3.(IY+d)	FD CB <u>d</u> 9E							23	
RES 4.A	CB A7							8	ビットリセット  ソースの第4ビット←0
RES 4.B	CB A0								
RES 4.C	CB A1								
RES 4.D	CB A2								
RES 4.E	CB A3								
RES 4.H	CB A4	-	-	-	-	-	-		
RES 4.L	CB A5								
RES 4.(HL)	CB A6							15	
RES 4.(IX+d)	DD CB <u>d</u> A6							23	
RES 4.(IY+d)	FD CB <u>d</u> A6							23	
RES 5.A	CB AF							8	ビットリセット  ソースの第5ビット←0
RES 5.B	CB A8								
RES 5.C	CB A9								
RES 5.D	CB AA								
RES 5.E	CB AB								
RES 5.H	CB AC	-	-	-	-	-	-		
RES 5.L	CB AD								
RES 5.(HL)	CB AE							15	
RES 5.(IX+d)	DD CB <u>d</u> AE							23	
RES 5.(IY+d)	FD CB <u>d</u> AE							23	

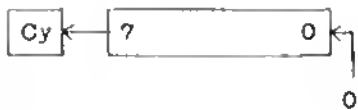
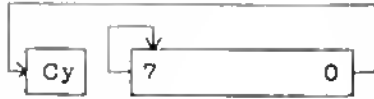
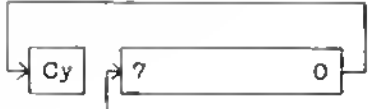
ニーモニック	マシン語	フ ラ グ 変 化							所 要 クロック サイクル	動 作
		S	Z	H	P/V		N	C		
RES 6.A	CB B7									ビットリセット  ソースの第6ビット ← 0
RES 6.B	CB B0									
RES 6.C	CB B1									
RES 6.D	CB B2								8	
RES 6.E	CB B3									
RES 6.H	CB B4									
RES 6.L	CB B5									
RES 6.(HL)	CB B6								15	
RES 6.(IX+d)	DD CB <u>d</u> B6								23	
RES 6.(IY+d)	FD CB <u>d</u> B6								23	
RES 7.A	CB BF									ビットリセット  ソースの第7ビット ← 0
RES 7.B	CB B8									
RES 7.C	CB B9									
RES 7.D	CB BA								8	
RES 7.E	CB BB									
RES 7.H	CB BC									
RES 7.L	CB BD									
RES 7.(HL)	CB BE								15	
RES 7.(IX+d)	DD CB <u>d</u> BE								23	
RES 7.(IY+d)	FD CB <u>d</u> BE								23	
RET	C9								10	サブルーチンからのリターン PC ヘスタックより POP
RET NZ	C0									条件付リターン ・ 条件が成立すれば POP PC ・ 不成立なら本命令は無視する
RET Z	C8									
RET NC	D0								条件成立 11	
RET C	D8								不成立 5	
RET PO	E0									
RET PE	E8									
RET P	F0									
RET M	F8									
RETI	ED 4D								14	割り込み処理からのリターン POP PC
RETN	ED 45								14	ノンマスカブル割り込み処理からのリターン POP PC
RLA	17			0			0		4	ローテート・レフト・アキュムレータ RL A と同じ
RLCA	07			0			0		4	ローテート・レフト・サーキュラ・アキュムレータ RLC A と同じ
RRA	1F			0			0		4	ローテート・ライト・アキュムレータ RR A と同じ
RRCA	0F			0			0		4	ローテート・ライト・サーキュラ・アキュムレータ RRC A と同じ

ニーモニック	マシン語	フ ラ グ 変 化						所 要 クロック サイクル	動 作
		S	Z	H	P/V P V	N	C		
RL A	CB 17							8	ローテート・レフト 
RL B	CB 10								
RL C	CB 11								
RL D	CB 12								
RL E	CB 13								
RL H	CB 14		0			0			
RL L	CB 15								
RL (HL)	CB 16								
RL (IX+d)	DD CB_d 16							23	
RL (IY+d)	FF CB_d 16							23	
RLC A	CB 07							8	ローテート・レフト・サーキュラ 
RLC B	CB 00								
RLC C	CB 01								
RLC D	CB 02								
RLC E	CB 03								
RLC H	CB 04		0			0			
RLC L	CB 05								
RLC (HL)	CB 06								
RLC (IX+d)	DD CB_d 06							23	
RLC (IY+d)	FD CB_d 06							23	
RR A	CB 1F							8	ローテート・ライト 
RR B	CB 18								
RR C	CB 19								
RR D	CB 1A								
RR E	CB 1B								
RR H	CB 1C		0			0			
RR L	CB 1D								
RR (HL)	CB 1E								
RR (IX+d)	DD CB_d 1E							23	
RR (IY+d)	FD CB_d 1E							23	
RRC A	CB 0F							8	ローテート・ライト・サーキュラ 
RRC B	CB 08								
RRC C	CB 09								
RRC D	CB 0A								
RRC E	CB 0B								
RRC H	CB 0C		0			0			
RRC L	CB 0D								
RRC (HL)	CB 0E								
RRC (IX+d)	DD CB_d 0E							23	
RRC (IY+d)	FD CB_d 0E							23	
RLD	ED 6F							18	ローテート・レフト・ディジット A 
RRD	ED 67							18	ローテート・ライト・ディジット A 



ニーモニック	マシン語	フラグ変化						所 要 クロック サイクル	動 作	
		S	Z	H	P/V		N			C
					P	V				
RST 00H	C7								リスタート  0000H-0038Hのいずれかの番地 に対するCALL	
RST 08H	CF									
RST 10H	D7									
RST 18H	DF									
RST 20H	E7	-	-	-	-	-	-	11		
RST 28H	EF									
RST 30H	F7									
RST 38H	FF									
SBC A, n	DE <u>n</u>							7	8ビット引き算(キャリ付) (サブトラクト・ウィズ・キャリ)  A ← A - ソース - Cy	
SBC A, A	9F									
SBC A, B	98									
SBC A, C	99									
SBC A, D	9A							4		
SBC A, E	9B	.	.	.	-	.	1	.		
SBC A, H	9C									
SBC A, L	9D									
SBC A, (HL)	9E							7		
SBC A, (IX+d)	DD 9E <u>d</u>							19		
SBC A, (IY+d)	FD 9E <u>d</u>							19		
SBC HL, BC	ED 42									16ビット引き算(キャリ付) (サブトラクト・ウィズ・キャリ) HL ← HL - ソース - Cy
SBC HL, DE	ED 52	.	.	x	-	.	1	.		
SBC HL, HL	ED 62									
SBC HL, SP	ED 72									
SCF	37	-	-	0	-	-	0	1	4	セット・キャリフラグ Cy ← 1
SET 0, A	CB C7								ビットセット  ソースの第0ビット ← 1	
SET 0, B	CB C0									
SET 0, C	CB C1									
SET 0, D	CB C2									
SET 0, E	CB C3									
SET 0, H	CB C4	-	-	-	-	-	-	-		
SET 0, L	CB C5									
SET 0, (HL)	CB C6							15		
SET 0, (IX+d)	DD CB <u>d</u> C6							23		
SET 0, (IY+d)	FD CB <u>d</u> C6							23		
SET 1, A	CB CF								ビットセット  ソースの第1ビット ← 1	
SET 1, B	CB C8									
SET 1, C	CB C9									
SET 1, D	CB CA									
SET 1, E	CB CB									
SET 1, H	CB CC	-	-	-	-	-	-	-		
SET 1, L	CB CD									
SET 1, (HL)	CB CE							15		
SET 1, (IX+d)	DD CB <u>d</u> CE							23		
SET 1, (IY+d)	FD CB <u>d</u> CE							23		

ニーモニック	マシン語	フ ラ グ 変 化							所 要 クロック サイクル	動 作
		S	Z	H	P/V		N	C		
					P	V				
SET 2.A	CB D7								8       15 23 23	ビットセット   ソースの第2ビット←1
SET 2.B	CB D0									
SET 2.C	CB D1									
SET 2.D	CB D2									
SET 2.E	CB D3									
SET 2.H	CB D4									
SET 2.L	CB D5									
SET 2.(HL)	CB D6									
SET 2.(IX+d)	DD CB <u>d</u> D6									
SET 2.(IY+d)	FD CB <u>d</u> D6									
SET 3.A	CB DF								8       15 23 23	ビットセット   ソースの第3ビット←1
SET 3.B	CB D8									
SET 3.C	CB D9									
SET 3.D	CB DA									
SET 3.E	CB DB									
SET 3.H	CB DC									
SET 3.L	CB DD									
SET 3.(HL)	CB DE									
SET 3.(IX+d)	DD CB <u>d</u> DE									
SET 3.(IY+d)	FD CB <u>d</u> DE									
SET 4.A	CB E7								8       15 23 23	ビットセット   ソースの第4ビット←1
SET 4.B	CB E0									
SET 4.C	CB E1									
SET 4.D	CB E2									
SET 4.E	CB E3									
SET 4.H	CB E4									
SET 4.L	CB E5									
SET 4.(HL)	CB E6									
SET 4.(IX+d)	DD CB <u>d</u> E6									
SET 4.(IY+d)	FD CB <u>d</u> E6									
SET 5.A	CB EF								8       15 23 23	ビットセット   ソースの第5ビット←1
SET 5.B	CB E8									
SET 5.C	CB E9									
SET 5.D	CB EA									
SET 5.E	CB EB									
SET 5.H	CB EC									
SET 5.L	CB ED									
SET 5.(HL)	CB EE									
SET 5.(IX+d)	DD CB <u>d</u> EE									
SET 5.(IY+d)	FD CB <u>d</u> EE									
SET 6.A	CB F7								8       15 23 23	ビットセット   ソースの第6ビット←1
SET 6.B	CB F0									
SET 6.C	CB F1									
SET 6.D	CB F2									
SET 6.E	CB F3									
SET 6.H	CB F4									
SET 6.L	CB F5									
SET 6.(HL)	CB F6									
SET 6.(IX+d)	DD CB <u>d</u> F6									
SET 6.(IY+d)	FD CB <u>d</u> F6									

ニーモニツク	マシン語	フ ラ グ 変 化							所 要 クロック サイクル	動 作
		S	Z	H	P/V		N	C		
					P	V				
SET 7, A	CB FF								8	ビットセット  ソースの第7ビット←1
SET 7, B	CB F8									
SET 7, C	CB F9									
SET 7, D	CB FA									
SET 7, E	CB FB									
SET 7, H	CB FC									
SET 7, L	CB FD									
SET 7, (HL)	CB FE								15	
SET 7, (IX+d)	DD CB <u>d</u> FE								23	
SET 7, (IY+d)	FD CB <u>d</u> FE								23	
SLA A	CB 27								8	シフト・レフト・アリスメチック  
SLA B	CB 20									
SLA C	CB 21									
SLA D	CB 22									
SLA E	CB 23									
SLA H	CB 24									
SLA L	CB 25									
SLA (HL)	CB 26								15	
SLA (IX+d)	DD CB <u>d</u> 26								23	
SLA (IY+d)	FD CB <u>d</u> 26								23	
SRA A	CB 2F								8	シフト・ライト・アリスメチック  
SRA B	CB 28									
SRA C	CB 29									
SRA D	CB 2A									
SRA E	CB 2B									
SRA H	CB 2C									
SRA L	CB 2D									
SRA (HL)	CB 2E								15	
SRA (IX+d)	DD CB <u>d</u> 2E								23	
SRA (IY+d)	FD CB <u>d</u> 2E								23	
SRL A	CB 3F								8	シフト・ライト・ロジカル  
SRL B	CB 38									
SRL C	CB 39									
SRL D	CB 3A									
SRL E	CB 3B									
SRL H	CB 3C									
SRL L	CB 3D									
SRL (HL)	CB 3E								15	
SRL (IX+d)	DD CB <u>d</u> 3E								23	
SRL (IY+d)	FD CB <u>d</u> 3E								23	
SUB n	D6 <u>n</u>								7	8ビット引き算 (サブトラクト)  A ← A - ソース
SUB A	97								4	
SUB B	90									
SUB C	91									
SUB D	92									
SUB E	93									
SUB H	94									
SUB L	95									
SUB (HL)	96								7	
SUB (IX+d)	DD 96 <u>d</u>								19	
SUB (IY+d)	FD 96 <u>d</u>								19	

ニーモニック	マシン語	フラグ変化							所 要 クロック サイクル	動 作															
		S	Z	H	P/V		N	C																	
					P	V																			
XOR n	EE <u>n</u>								7	排他的論理和(エクスクルーシブ・オア)  A ← A ⊕ ソース <table border="1"><tr><td>a</td><td>b</td><td>答</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	a	b	答	0	0	0	0	1	1	1	0	1	1	1	0
a	b	答																							
0	0	0																							
0	1	1																							
1	0	1																							
1	1	0																							
XOR A	AF																								
XOR B	A8																								
XOR C	A9																								
XOR D	AA								4																
XOR E	AB			0			0	0																	
XOR H	AC																								
XOR L	AD																								
XOR (HL)	AE								7																
XOR (IX+d)	DD AE <u>d</u>								19																
XOR (IY+d)	FD AE <u>d</u>								19																

## フ ラ グ 変 化 表

	命	令	フ ラ グ							コ メ ン ト
			S	Z	H	P/V		N	C	
						P	V			
1	8ビット加算	ADD ADC	・	・	・	—	・	0	・	Cフラグを0にするとき
2	8ビット減算系	SUB SBC CP NEG	・	・	・	—	・	1	・	
3	論理積	AND	・	・	1	・	—	0	0	
4	論理和	OR XOR	・	・	0	・	—	0	0	
5	8ビットインクリメント	INC	・	・	・	—	・	0	—	16ビットINC DEC では変化しない
6	8ビットデクリメント	DEC	・	・	・	—	・	1	—	
7	16ビット加算	ADD	—	—	×	—	—	0	・	
8	16ビットキャリ付加算	ADC	・	・	×	—	・	0	・	
9	16ビットキャリ付減算	SBC	・	・	×	—	・	1	・	
10	ローテート・アキュムレータ	RLA RLCA RRA RRCA	—	—	0	—	—	0	・	
11	ローテート	RL RLC RR RRC	・	・	0	・	—	0	・	
12	シフト	SLA SRA SRL	・	・	0	・	—	0	・	
13	ローテート・ディジット	RLD RRD	・	・	0	・	—	0	—	
14	10進補正	DAA	・	・	・	・	—	—	・	
15	ビット反転	CPL	—	—	1	—	—	1	—	
16	セットキャリ	SCF	—	—	0	—	—	0	1	
17	キャリ反転	CCF	—	—	×	—	—	0	・	
18	間接アドレッシング入力	IN r, (C)	・	・	0	・	—	0	—	IN A.(n)では 変化しない
19	ブロック入力出力	INI IND OUTI OUTD	×	・	×	×	—	×	×	
20	リビート入力出力	INIR INDR OTIR OTDR	×	1	×	×	—	×	×	
21	ブロック転送	LDI LDD	×	×	0	—	・	0	—	
22	リビート転送	LDIR LDDR	×	×	0	—	0	0	—	
23	ブロックサーチ	CPI CPD CPIR CPDR	×	・	×	—	・	1	—	
24	Iレジスタ, Rレジスタ	LD A, I LD A, R	・	・	0	IFF	0	—	P/VにIFFが コピーされる	
25	ビットテスト	BIT	×	・	1	×	—	0	—	

× 不定

1 1になる

0 0になる

・ 状態にしたがってセット, リセットされる

— 変化せず

IFF: 0のとき割り込み禁止(DI)

1のとき割り込み可(EI)

になっている

LD A, I LD A, R

ではこの値がP/Vにコピーされる

## 付 2 Z-80 規 格 表 (参考)

## (1) CPU

## 絶対最大定格

項 目	記 号	定 格 値	単 位
入 力 電 圧	$V_{IN}$	$-0.3 \sim +7$	V
出 力 電 圧	$V_{OUT}$	$-0.3 \sim +7$	V
動 作 温 度	$T_{opr}$	$0 \sim +70$	℃
保 存 温 度	$T_{stg}$	$-65 \sim +150$	℃

## 電気的特性

## DC 特性

 $(T_a = 0^\circ\text{C} \sim +70^\circ\text{C}, V_{CC} = +5\text{V} \pm 5\%)$ 

記 号	項 目	最 小 値	最 大 値	単 位	測 定 条 件
$V_{ILC}$	クロック "L" 入力電圧	-0.3	0.45	V	
$V_{IHC}$	クロック "H" 入力電圧	$V_{CC}-0.6$	$V_{CC}+0.3$	V	
$V_{IL}$	"L" 入力電圧	-0.3	0.8	V	
$V_{IH}$	"H" 入力電圧	2.0	$V_{CC}$	V	
$V_{OL}$	"L" 出力電圧		0.4	V	$I_{OL} = 1.8\text{mA}$
$V_{OH}$	"H" 出力電圧	2.4		V	$I_{OH} = -250\mu\text{A}$
$I_{CC}$	消費電流		150 200	mA	(上段 Z-80 CPU 下段 Z-80A CPU)
$I_{LI}$	入力リーク電流		10	$\mu\text{A}$	$V_{IN} = 0 \sim V_{CC}$
$I_{LOH}$	トライステート出力リーク電流		10	$\mu\text{A}$	$V_{OUT} = 2.4\text{V} \sim V_{CC}$
$I_{LOL}$	トライステート出力リーク電流		-10	$\mu\text{A}$	$V_{OUT} = 0.4\text{V}$
$I_{LD}$	入力時のデータ・バスのリーク電流		$\pm 10$	$\mu\text{A}$	$0 \leq V_{IN} \leq V_{CC}$

## 端子容量

 $(T_a = +25^\circ\text{C}, f = 1\text{MHz})$ 

記 号	項 目	最大値	単 位	測 定 条 件
$C_\Phi$	クロック入力容量	50	pF	被測定端子以外のすべての端子は接地
$C_{IN}$	入力容量	8	pF	
$C_{OUT}$	出力容量	12	pF	

## AC 特性

(Ta = 0°C ~ +70°C, V<sub>CC</sub> = +5V ± 5%)

信号	記号	パラメータ	Z-80 CPU		Z-80A CPU		単位	測定条件
			最小値	最大値	最小値	最大値		
Φ	t <sub>C</sub>	クロック周期	0.4	200	0.25	200	μs	
	t <sub>w</sub> (ΦH)	クロック・パルス幅("H")	180		110		ns	
	t <sub>w</sub> (ΦL)	クロック・パルス幅("L")	180	2000	110	2000	ns	
	t <sub>raif</sub>	クロックの立ち上がり・立ち下がり時間		30		30	ns	
A <sub>0</sub> -A <sub>15</sub>	t <sub>D</sub> (AD)	クロックの立ち上がりから出力までの遅延		145		110	ns	C <sub>L</sub> = 50pF
	t <sub>F</sub> (AD)	出力がフロート状態になるまでの遅延		110		90	ns	
	t <sub>acm</sub>	MREQ に先立つ出力確定時間 (メモリ・サイクル)	[1]		[1]		ns	
	t <sub>aci</sub>	I/O <sub>RQ</sub> , RD または WR に先立つ出力確定時間 (入出力サイクル)	[2]		[2]		ns	
	t <sub>ca</sub>	RD, WR, I/O <sub>RQ</sub> または MREQ からの出力保持時間	[3]		[3]		ns	
	t <sub>caf</sub>	RD または WR からの出力保持時間 (フロート状態への遷移時)	[4]		[4]		ns	
D <sub>0</sub> -D <sub>7</sub>	t <sub>D</sub> (D)	クロックの立ち下がりから出力までの遅延		230		150	ns	C <sub>L</sub> = 50pF
	t <sub>F</sub> (D)	出力がフロート状態になるまでの遅延 (書き込みサイクル)		90		90	ns	
	t <sub>sΦ</sub> (D)	クロックの立ち上がりに対するセットアップ時間 (M1 サイクル)	50		35		ns	
	t <sub>sΦ</sub> (D)	クロックの立ち下がりに対するセットアップ時間 (M2 ~ M5 サイクル)	60		50		ns	
	t <sub>dcm</sub>	WR に先立つ出力確定時間 (メモリ・サイクル)	[5]		[5]		ns	
	t <sub>dci</sub>	WR に先立つ出力確定時間 (入出力サイクル)	[6]		[6]		ns	
	t <sub>cdf</sub>	WR からの出力保存時間	[7]		[7]		ns	
	t <sub>H</sub>	ホールド時間	0		0		ns	
MREQ	t <sub>DLΦ</sub> (MR)	クロックの立ち下がりから MREQ = "L" になるまでの遅延		100		85	ns	C <sub>L</sub> = 50pF
	t <sub>DHΦ</sub> (MR)	クロックの立ち上がりから MREQ = "H" になるまでの遅延 (M1 サイクル)		100		85	ns	

信号	記号	パラメータ	Z-80 CPU		Z-80A CPU		単位	測定条件
			最小値	最大値	最小値	最大値		
$\overline{\text{MREQ}}$	$t_{\text{DH}}(\overline{\text{MR}})$	クロックの立ち下がりから $\overline{\text{MREQ}} = "H"$ になるまでの遅延 (M2 ~ M5 サイクル)		100		85	ns	$C_L = 50 \text{ pF}$
	$t_w(\overline{\text{MR}}_L)$	$\overline{\text{MREQ}}$ のパルス幅 ("L")	[8]		[8]		ns	
	$t_w(\overline{\text{MR}}_H)$	$\overline{\text{MREQ}}$ のパルス幅 ("H")	[9]		[9]		ns	
$\overline{\text{IORQ}}$	$t_{\text{DL}}(\overline{\text{IR}})$	クロックの立ち上がりから $\overline{\text{IORQ}} = "L"$ になるまでの遅延 (入出力サイクル)		90		75	ns	$C_L = 50 \text{ pF}$
	$t_{\text{DL}}(\overline{\text{IR}})$	クロックの立ち下がりから $\overline{\text{IORQ}} = "L"$ になるまでの遅延 (INTA サイクル)		110		85	ns	
	$t_{\text{DH}}(\overline{\text{IR}})$	クロックの立ち上がりから $\overline{\text{IORQ}} = "H"$ になるまでの遅延 (INTA サイクル)		100		85	ns	
	$t_{\text{DH}}(\overline{\text{IR}})$	クロックの立ち下がりから $\overline{\text{IORQ}} = "H"$ になるまでの遅延 (入出力サイクル)		110		85	ns	
$\overline{\text{RD}}$	$t_{\text{DL}}(\overline{\text{RD}})$	クロックの立ち上がりから $\overline{\text{RD}} = "L"$ になるまでの遅延 (入出力サイクル)		100		85	ns	$C_L = 50 \text{ pF}$
	$t_{\text{DL}}(\overline{\text{RD}})$	クロックの立ち下がりから $\overline{\text{RD}} = "L"$ になるまでの遅延 (メモリ・サイクル)		130		95	ns	
	$t_{\text{DH}}(\overline{\text{RD}})$	クロックの立ち上がりから $\overline{\text{RD}} = "H"$ になるまでの遅延 (M1 サイクル)		100		85	ns	
	$t_{\text{DH}}(\overline{\text{RD}})$	クロックの立ち下がりから $\overline{\text{RD}} = "H"$ になるまでの遅延 (M2 ~ M5 サイクル)		110		85	ns	
$\overline{\text{WR}}$	$t_{\text{DL}}(\overline{\text{WR}})$	クロックの立ち上がりから $\overline{\text{WR}} = "L"$ になるまでの遅延 (入出力サイクル)		80		65	ns	$C_L = 50 \text{ pF}$
	$t_{\text{DL}}(\overline{\text{WR}})$	クロックの立ち下がりから $\overline{\text{WR}} = "L"$ になるまでの遅延 (メモリ・サイクル)		90		80	ns	
	$t_{\text{DH}}(\overline{\text{WR}})$	クロックの立ち下がりから $\overline{\text{WR}} = "H"$ になるまでの遅延		100		80	ns	
	$t_w(\overline{\text{WR}}_L)$	$\overline{\text{WR}}$ のパルス幅 ("L")	[10]		[10]		ns	
$\overline{\text{M1}}$	$t_{\text{DL}}(\overline{\text{M1}})$	クロックの立ち上がりから $\overline{\text{M1}} = "L"$ になるまでの遅延		130		100	ns	$C_L = 50 \text{ pF}$
	$t_{\text{DH}}(\overline{\text{M1}})$	クロックの立ち上がりから $\overline{\text{M1}} = "H"$ になるまでの遅延		130		100	ns	
$\overline{\text{RFSH}}$	$t_{\text{DL}}(\overline{\text{RF}})$	クロックの立ち上がりから $\overline{\text{RFSH}} = "L"$ になるまでの遅延		180		130	ns	$C_L = 50 \text{ pF}$



信号	記号	パラメータ	Z-80 CPU		Z-80A CPU		単位	測定条件
			最小値	最大値	最小値	最大値		
$\overline{\text{RFSH}}$	$t_{\text{DH}}(\text{RF})$	クロックの立ち上がりから $\overline{\text{RFSH}} = "H"$ になるまでの遅延		150		120	ns	$C_L = 50 \text{ pF}$
$\overline{\text{WAIT}}$	$t_s(\text{WT})$	クロックの立ち下がりに対するセットアップ時間	70		70		ns	
$\overline{\text{HALT}}$	$t_D(\text{HT})$	クロックの立ち下がりからの遅延		300		300	ns	$C_L = 50 \text{ pF}$
$\overline{\text{INT}}$	$t_s(\text{IT})$	クロックの立ち上がりに対するセットアップ時間	80		80		ns	
$\overline{\text{NMI}}$	$t_w(\text{NMI})$	$\overline{\text{NMI}}$ のパルス幅("L")	80		80		ns	
$\overline{\text{BUSRQ}}$	$t_s(\text{BQ})$	クロックの立ち上がりに対するセットアップ時間	80		50		ns	
$\overline{\text{BUSAK}}$	$t_{\text{DL}}(\text{BA})$	クロックの立ち上がりから $\overline{\text{BUSAK}} = "L"$ になるまでの遅延		120		100	ns	$C_L = 50 \text{ pF}$
	$t_{\text{DH}}(\text{BA})$	クロックの立ち下がりから $\overline{\text{BUSAK}} = "H"$ になるまでの遅延		110		100	ns	
$\overline{\text{RESET}}$	$t_s(\text{RS})$	クロックの立ち上がりに対するセットアップ時間	90		60		ns	
	$t_f(\text{C})$	フロート状態になるまでの遅延 ( $\overline{\text{MREQ}}$ , $\overline{\text{IORQ}}$ , $\overline{\text{RD}}$ および $\overline{\text{WR}}$ )		100		80	ns	
	$t_{\text{mr}}$	$\overline{\text{IORQ}}$ に先立つ $\overline{\text{M1}}$ 出力("L")の確定時間 ( $\overline{\text{INTA}}$ サイクル)	[11]		[11]		ns	

注 [1]  $t_{\text{acm}} = t_w(\Phi_H) + t_f - 75$

[2]  $t_{\text{aci}} = t_c - 80$

[3]  $t_{\text{ca}} = t_w(\Phi_L) + t_r - 40$

[4]  $t_{\text{caf}} = t_w(\Phi_L) + t_r - 60$

[5]  $t_{\text{dcm}} = t_c - 210$

[6]  $t_{\text{dei}} = t_w(\Phi_L) + t_r - 210$

[7]  $t_{\text{cdf}} = t_w(\Phi_L) + t_r - 80$

[8]  $t_w(\overline{\text{MR}}_L) = t_c - 40$

[9]  $t_w(\overline{\text{MR}}_H) = t_w(\Phi_H) + t_f - 30$

[10]  $t_w(\overline{\text{WR}}_L) = t_c - 40$

[11]  $t_{\text{mr}} = 2t_c + t_w(\Phi_H) + t_f - 80$

○データを  $\overline{\text{RD}}$  に同期してバスに送り出すことが望ましい。割り込みアクノリッジ・サイクルでは  $\overline{\text{M1}}$  および  $\overline{\text{IORQ}}$  の両方に同期して送り出すことが望ましい。

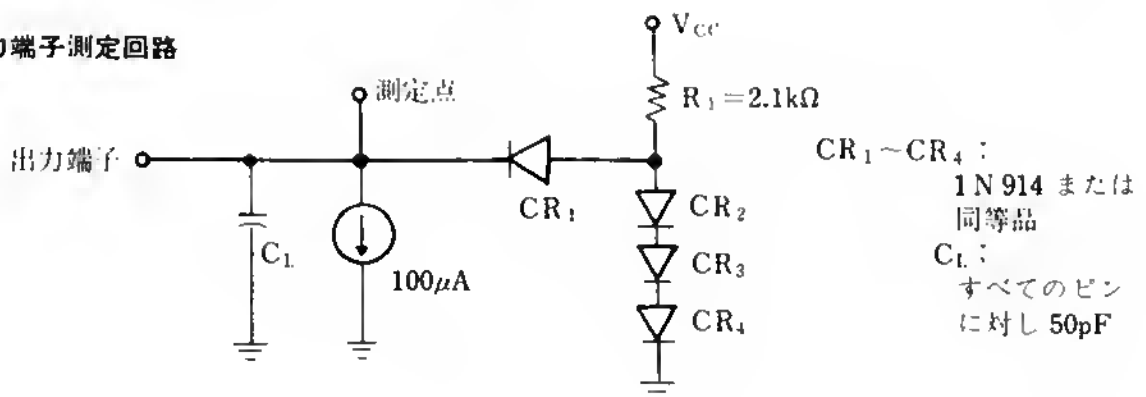
○制御信号はすべて内部で同期がとれているため、クロックについて非同期的に使用してもよい。

○ $T_a = +70^\circ\text{C}$ ,  $V_{\text{CC}} = +5\text{V} \pm 5\%$  における負荷容量と出力の遅延との関係は次のとおりです。

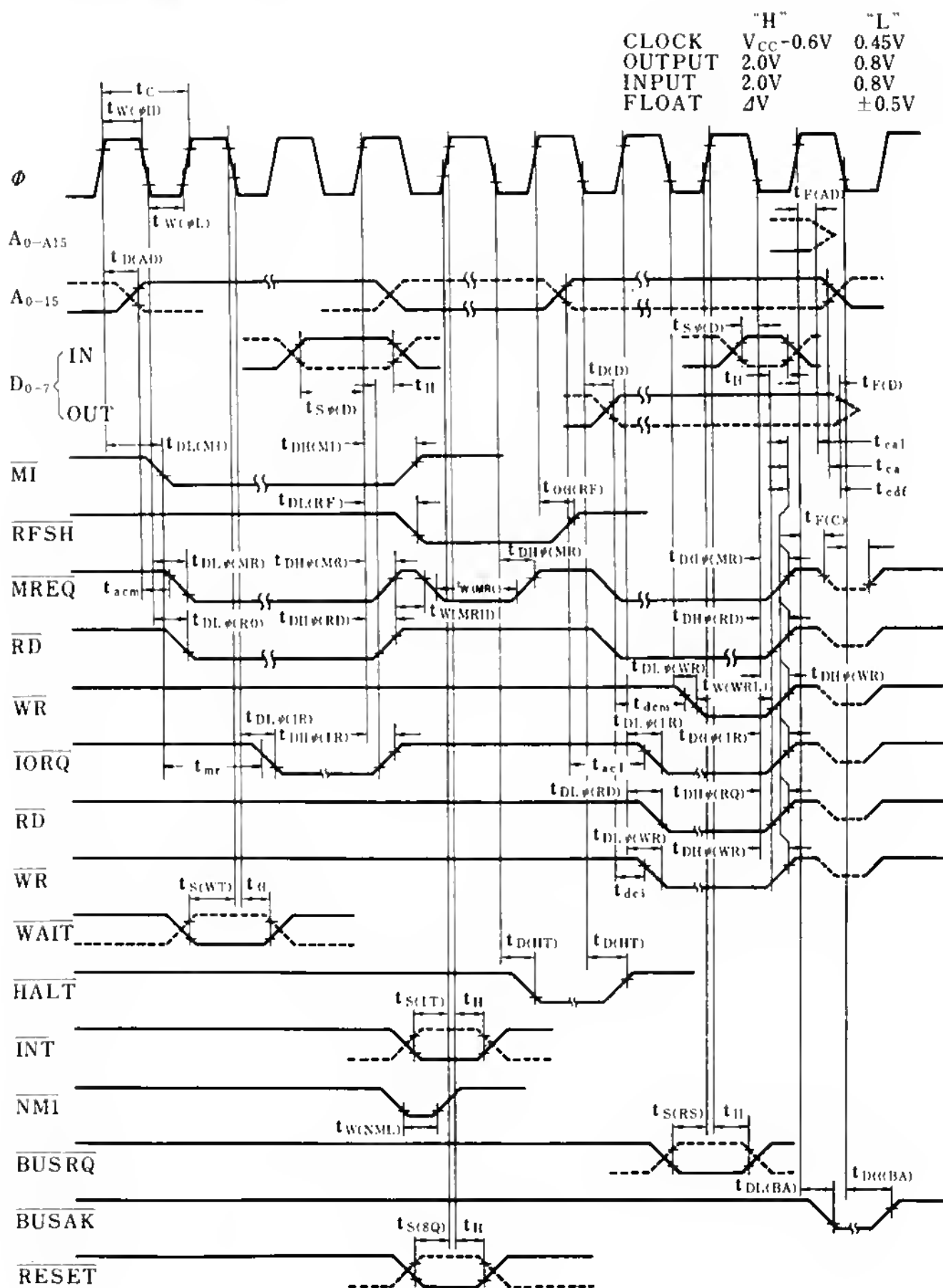
負荷容量の  $50\text{pF}$  増加につき遅延は  $10\text{ns}$  増加します。負荷容量の最大値は、データ・バスが  $200\text{pF}$  で、他は  $100\text{pF}$  です。

○ $\overline{\text{RESET}}$  の入力幅は最低 3 クロック・サイクル必要です。

出力端子測定回路



### AC タイミング図



## ( 2 ) PIO

## 絶対最大定格

項 目	記 号	定 格 値	単 位
入 力 電 圧	$V_{IN}$	$-0.3 \sim +7$	V
出 力 電 圧	$V_{OUT}$	$-0.3 \sim +7$	V
動 作 温 度	$T_{opr}$	$0 \sim +70$	℃
保 存 温 度	$T_{stg}$	$-65 \sim +150$	℃

## 電気的特性

## DC 特性

 $(T_a = 0^\circ\text{C} \sim +70^\circ\text{C}, V_{CC} = +5\text{V} \pm 5\%)$ 

記 号	項 目	最 小 値	最 大 値	単 位	測 定 条 件
$V_{ILC}$	クロック“L”入力電圧	-0.3	0.45	V	
$V_{IHC}$	クロック“H”入力電圧	$V_{CC} - 0.6$	$V_{CC} + 0.3$	V	
$V_{IL}$	“L”入力電圧	-0.3	0.8	V	
$V_{IH}$	“H”入力電圧	2.0	$V_{CC}$	V	
$V_{OL}$	“L”出力電圧		0.4	V	$I_{OL} = 2\text{mA}$
$V_{OH}$	“H”出力電圧	2.4		V	$I_{OH} = -250\mu\text{A}$
$I_{CC}$	消費電流		70	mA	
$I_{LI}$	入力リーク電流		10	$\mu\text{A}$	$V_{IN} = 0 \sim V_{CC}$
$I_{LOH}$	トライステート出力リーク電流		10	$\mu\text{A}$	$V_{OUT} = 2.4\text{V} \sim V_{CC}$
$I_{LOL}$	トライステート出力リーク電流		-10	$\mu\text{A}$	$V_{OUT} = 0.4\text{V}$
$I_{LD}$	入力時のデータ・バスのリーク電流		$\pm 10$	$\mu\text{A}$	$0 \leq V_{IN} \leq V_{CC}$
$I_{OHD}$	ダーリントン駆動電流	-1.5		mA	$V_{OH} = 1.5\text{V}$ ポートBのみ

## 端子容量

 $(T_a = +25^\circ\text{C}, f = 1\text{MHz})$ 

記 号	項 目	最大値	単 位	測 定 条 件
$C_\phi$	クロック入力容量	12	pF	被測定端子以外のすべての端子は接地
$C_{IN}$	入力容量	7	pF	
$C_{OUT}$	出力容量	10	pF	

## AC 特性

(T<sub>a</sub> = 0°C ~ +70°C, V<sub>CC</sub> = +5V ± 5%)

信号	記号	パラメータ	Z-80 PIO		Z-80A PIO		単位	測定条件
			最小値	最大値	最小値	最大値		
Φ	t <sub>C</sub>	クロック周期	400	[1]	250	[1]	ns	
	t <sub>W</sub> (中H)	クロック・パルス幅 ("H")	170	2000	105	2000	ns	
	t <sub>W</sub> (中L)	クロック・パルス幅 ("L")	170	2000	105	2000	ns	
	t <sub>r</sub> , t <sub>f</sub>	クロック立ち上がり・立ち下がり時間		30		30	ns	
	t <sub>H</sub>	ホールド時間	0		0		ns	
$\overline{CE}$ , C/D, B/A	t <sub>SD</sub> (CS)	読み出しまたは書き込みサイクルの制御信号のセットアップ時間	280		145		ns	
D <sub>0</sub> - D <sub>7</sub>	t <sub>DR</sub> (D)	$\overline{RD}$ の立ち下がりからデータ出力までの遅延		430 [2]		380 [2]	ns	C <sub>L</sub> = 50pF
	t <sub>SD</sub> (D)	書き込みまたはM1サイクルのデータのセットアップ時間	50		50		ns	
	t <sub>DI</sub> (D)	INTAサイクルの $\overline{IORQ}$ の立ち下がりからデータ出力までの遅延		340 [2]		250 [2]	ns	
	t <sub>F</sub> (D)	$\overline{RD}$ または $\overline{IORQ}$ の立ち上がりから出力バッファ・フロートまでの遅延		160		110	ns	
IEI	t <sub>S</sub> (IEI)	INTAサイクルの $\overline{IORQ}$ の立ち下がりに対するセットアップ時間	140		140		ns	
IEO	t <sub>DI</sub> (IO)	IEIの立ち上がりからの遅延		210 [4]		160 [4]	ns	C <sub>L</sub> = 50pF
	t <sub>DI</sub> (IO)	IEIの立ち下がりからの遅延 (注1)		190 [4]		130 [4]	ns	
	t <sub>DM</sub> (IO)	$\overline{MI}$ の立ち下がりからの遅延 (M1サイクルの直前で割り込みが発生したとき)		300 [4]		190 [4]	ns	
$\overline{IORQ}$	t <sub>SD</sub> (IR)	読み出しまたは書き込みサイクルのセットアップ時間	250		115		ns	
$\overline{MI}$	t <sub>SD</sub> (M1)	INTAまたはM1サイクルのセットアップ時間	210		90		ns	
$\overline{RD}$	t <sub>SD</sub> (RD)	読み出しまたはM1サイクルのセットアップ時間	240		115		ns	
$\overline{INT}$	t <sub>D</sub> (IT)	$\overline{STB}$ の立ち上がりからの遅延		490		440	ns	
	t <sub>D</sub> (IT <sub>3</sub> )	モード3のときのデータ一致からの遅延		420		380	ns	

信号	記号	パラメータ	Z-80 PIO		Z-80A PIO		単位	測定条件
			最小値	最大値	最小値	最大値		
A <sub>0</sub> →A <sub>7</sub> , B <sub>0</sub> →B <sub>7</sub>	t <sub>S</sub> (PD)	モード1のときの $\overline{\text{STB}}$ の立ち上がりに対するセットアップ時間	260		230		ns	C <sub>L</sub> =50pF
	t <sub>DS</sub> (PD)	モード2のときの $\overline{\text{STB}}$ の立ち下がりに対するセットアップ時間		230 [4]		210 [4]	ns	
	t <sub>F</sub> (PD)	モード2のときの $\overline{\text{STB}}$ の立ち上がりからポート・バス・フロートまでの遅延		200		180	ns	
	t <sub>DI</sub> (PD)	モード0のときの書き込みサイクルの $\overline{\text{IORQ}}$ の立ち上がりからポート出力確定までの遅延		200 [4]		180 [4]	ns	
A $\overline{\text{STB}}$ B $\overline{\text{STB}}$	t <sub>w</sub> (ST)	$\overline{\text{STB}}$ のパルス幅 ("L")	150 [3]		150 [3]		ns	
A RDY,	t <sub>DH</sub> (RY)	$\overline{\text{IORQ}}$ の立ち上がりからの応答時間		t <sub>c</sub> +480 [4]		t <sub>c</sub> +410 [4]	ns	C <sub>L</sub> =50pF
B RDY	t <sub>DL</sub> (RY)	$\overline{\text{STB}}$ の立ち上がりからの応答時間		t <sub>c</sub> +400 [4]		t <sub>c</sub> +360 [4]	ns	

注 (1)  $t_c = t_w(\Phi H) + t_w(\Phi L) + t_f + t_r$

(2) 負荷容量の50pF増加につき、遅延は10ns増加します。負荷容量の最大値は200pFです。

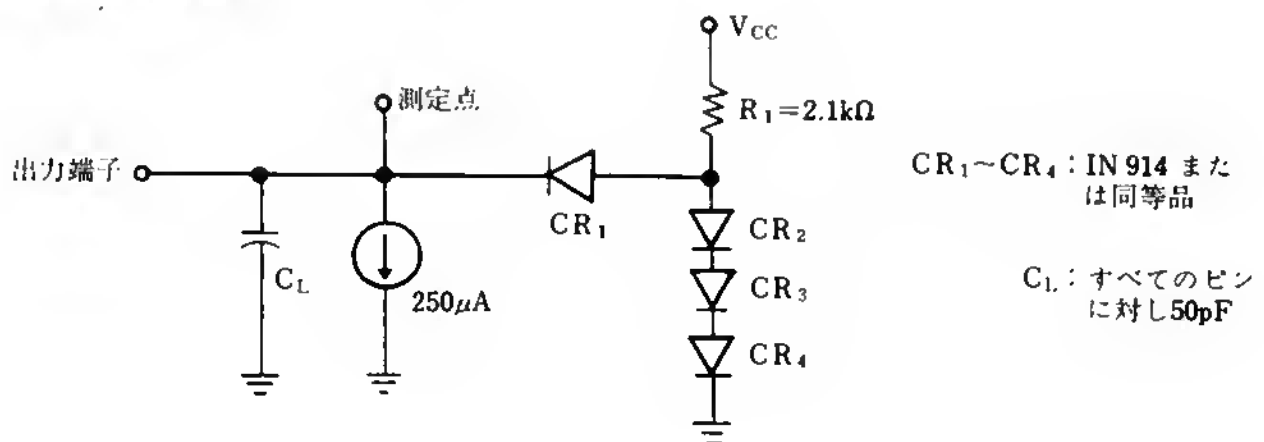
(3) モード2のときは、 $t_w(\text{ST}) > t_s(\text{PD})$ となります。

(4) 負荷容量の10pF増加につき、遅延は2ns増加します。負荷容量の最大値は100pFです。

(注I) デーザー・チェーンがN段ある場合

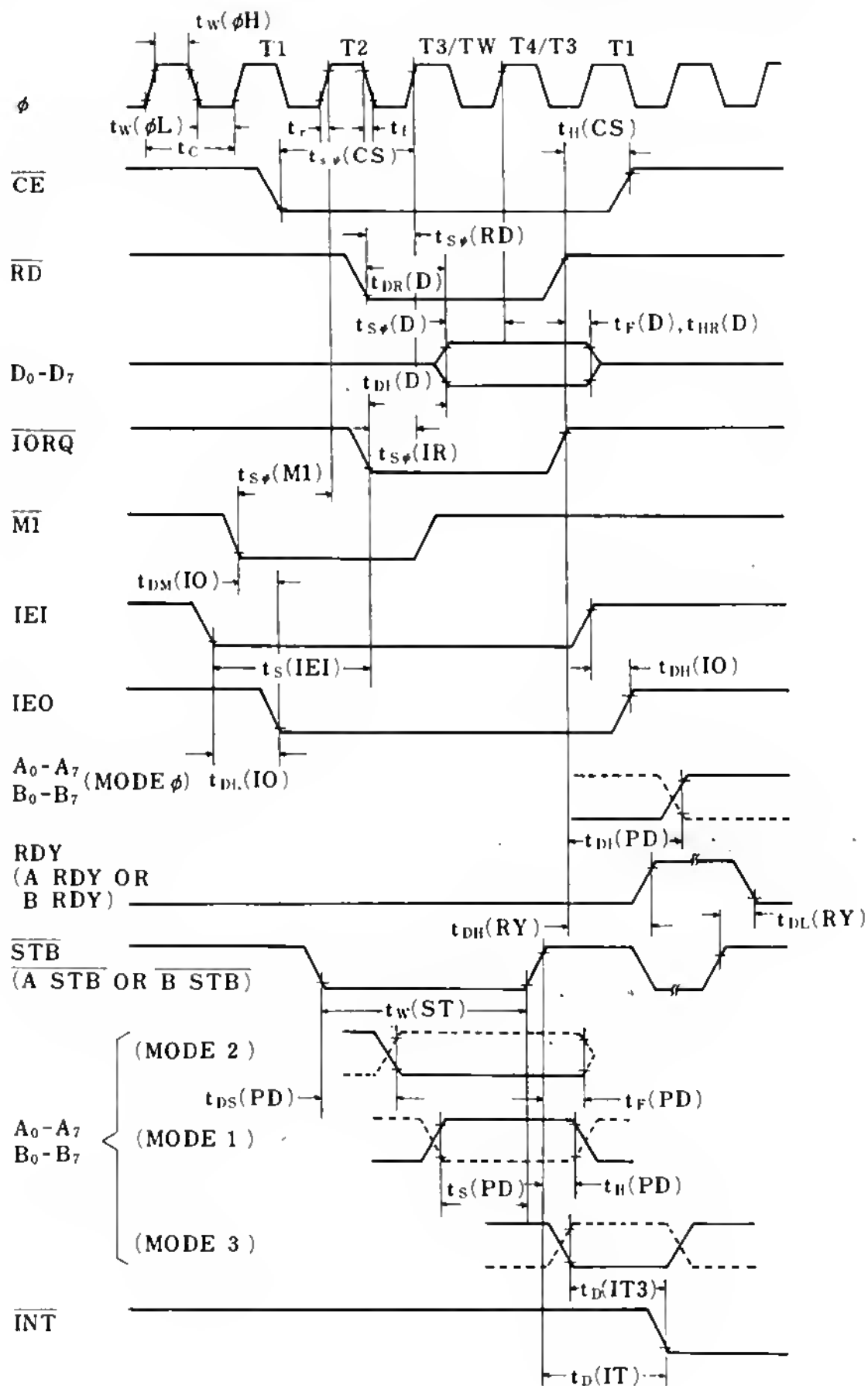
$2.5t_c > (N-2)t_{DL}(\text{IO}) + t_{DM}(\text{IO}) + t_s(\text{IEI}) + \text{TTLバッファ遅延}$ を満たさなければなりません。

## 出力端子測定回路



## ACタイミング図

	"H"	"L"
CLOCK	4.2V	0.8V
OUTPUT	2.0V	0.8V
INPUT	2.0V	0.8V
FLOAT	$\Delta V$	+0.5V



## ( 3 ) CTC

## 絶対最大定格

項 目	記 号	定 格	単位
入 力 電 圧	$V_{IN}$	$-0.3 \sim + 7$	V
出 力 電 圧	$V_{OUT}$	$-0.3 \sim + 7$	V
動 作 温 度	$T_{opr}$	$0 \sim + 70$	℃
保 存 温 度	$T_{stg}$	$-65 \sim +150$	℃

## 電気的特性

## DC 特性

(  $T_a = 0^\circ\text{C} \sim +70^\circ\text{C}$  ,  $V_{CC} = +5\text{V} \pm 5\%$  )

記 号	項 目	最 小 値	最 大 値	単位	測 定 条 件
$V_{ILC}$	クロック “L” 入力電圧	-0.3	0.45	V	
$V_{IHC}$	クロック “H” 入力電圧	$V_{CC}-0.6$	$V_{CC}+0.3$	V	
$V_{IL}$	“L” 入力電圧	-0.3	0.8	V	
$V_{IH}$	“H” 入力電圧	2.0	$V_{CC}$	V	
$V_{OL}$	“L” 出力電圧		0.4	V	$I_{OL} = 2\text{mA}$
$V_{OH}$	“H” 出力電圧	2.4		V	$I_{OH} = -250\mu\text{A}$
$I_{CC}$	消費電流		120	mA	$t_C = 400\text{ns}$
$I_{LI}$	入力リーク電流		10	$\mu\text{A}$	$V_{IN} = 0\text{V} \sim V_{CC}$
$I_{LOH}$	トライステート出力リーク電流		10*	$\mu\text{A}$	$V_{OUT} = 2.4\text{V} \sim V_{CC}$
$I_{LOL}$	トライステート出力リーク電流		-10*	$\mu\text{A}$	$V_{OUT} = 0.4\text{V}$
$I_{OHD}$	ダーリントン駆動電流	-1.5*		mA	$V_{OH} = 1.5\text{V}$ $ZC/TO_0 \sim ZC/TO_2$ に適用

\* 流入電流を正、流出電流を負とします。

## 端子容量

(  $T_a = +25^\circ\text{C}$  ,  $f = 1\text{MHz}$  )

記 号	項 目	最大値	単 位	測 定 条 件
$C_\Phi$	クロック入力容量	25	pF	被測定端子以外のすべての端子は接地
$C_{IN}$	入 力 容 量	5	pF	
$C_{OUT}$	出 力 容 量	10	pF	

## AC 特性

(T<sub>a</sub> = 0 °C ~ +70 °C、V<sub>CC</sub> = +5 V ± 5 %)

信号	記号	パラメータ	Z-80 CTC		Z-80A CTC		単位	備考
			最小値	最大値	最小値	最大値		
Φ	t <sub>C</sub>	クロック周期	400	[1]	250	[1]	ns	
	t <sub>W(ΦH)</sub>	クロック・パルス幅("H")	170	2000	105	2000	ns	
	t <sub>W(ΦL)</sub>	クロック・パルス幅("L")	170	2000	105	2000	ns	
	t <sub>RI, IF</sub>	クロック立ち上がり・立ち下がり時間		30		30	ns	
	t <sub>H</sub>	ホールド時間	0		0		ns	
CS, $\overline{CE}$	t <sub>SP(CS)</sub>	読み出し、または書き込みサイクルの制御信号のセットアップ時間	160		60		ns	
D <sub>0</sub> - D <sub>7</sub>	t <sub>DR(D)</sub>	$\overline{RD}$ の立ち下がりからデータ出力までの遅延		480		380	ns	[2]
	t <sub>SP(D)</sub>	書き込み、またはM1サイクルのデータのセットアップ時間	60		50		ns	
	t <sub>DI(D)</sub>	INTAサイクルの $\overline{IORQ}$ の立ち下がりからデータ出力までの遅延		340		160	ns	[2]
	t <sub>F(D)</sub>	$\overline{RD}$ の立ち上がりから出力バッファ・フロートまでの遅延		230		110	ns	
IEI	t <sub>S(IEI)</sub>	INTAサイクルの $\overline{IORQ}$ の立ち下がりに対するセットアップ時間	200		140		ns	
IEO	t <sub>DH(IEO)</sub>	IEIの立ち上がりからの遅延		220		160	ns	[3]
	t <sub>DL(IEO)</sub>	IEIの立ち下がりからの遅延		190		130	ns	[3]
	t <sub>DM(IEO)</sub>	M1の立ち下がりからの遅延 (M1サイクルの直前で割り込みが発生したとき)		300		190	ns	[3]
$\overline{IORQ}$	t <sub>SP(1R)</sub>	読み出し、または書き込みサイクルのセットアップ時間	250		115		ns	
$\overline{M1}$	t <sub>SP(M1)</sub>	INTA、またはM1サイクルのセットアップ時間	210		90		ns	
$\overline{RD}$	t <sub>SP(RD)</sub>	読み出し、またはM1サイクルのセットアップ時間	240		115		ns	
$\overline{INT}$	t <sub>DCX(1T)</sub>	CLK/TRGの立ち上がりからの遅延		2t <sub>C(Φ)</sub> + 200		2t <sub>C(Φ)</sub> + 140	ns	カウンタ・モード
	t <sub>DΦ(1T)</sub>	Φの立ち上がりからの遅延		t <sub>C(Φ)</sub> + 200		2t <sub>C(Φ)</sub> + 140	ns	タイマ・モード
CLK/TRG 0-3	t <sub>C(CK)</sub>	カウンタ・クロック周期		2t <sub>C(Φ)</sub>		2t <sub>C(Φ)</sub>	ns	カウンタ・モード
	t <sub>F(CK/TR)</sub> t <sub>F(CK/TR)</sub>	カウンタ・クロックおよびトリガの立ち上がり・立ち下がり時間		50		30	ns	
	t <sub>S(CK)</sub>	即時カウントに要するクロックのセットアップ時間	210		130		ns	カウンタ・モード
	t <sub>S(TR)</sub>	プリスケアラの即時起動に要するトリガのセットアップ時間	210		130		ns	タイマ・モード
	t <sub>W(CTH)</sub>	カウンタ・クロックおよびトリガのパルス幅("H")	200		120		ns	カウンタ・モード および
	t <sub>W(CTL)</sub>	カウンタ・クロックおよびトリガのパルス幅("L")	200		120		ns	タイマ・モード
ZC/T0 0-2	t <sub>DH(ZC)</sub>	Φの立ち上がりからZC/T0="H"までの遅延		190		120	ns	カウンタ・モード および
	t <sub>DL(ZC)</sub>	Φの立ち下がりからZC/T0="L"までの遅延		190		120	ns	タイマ・モード



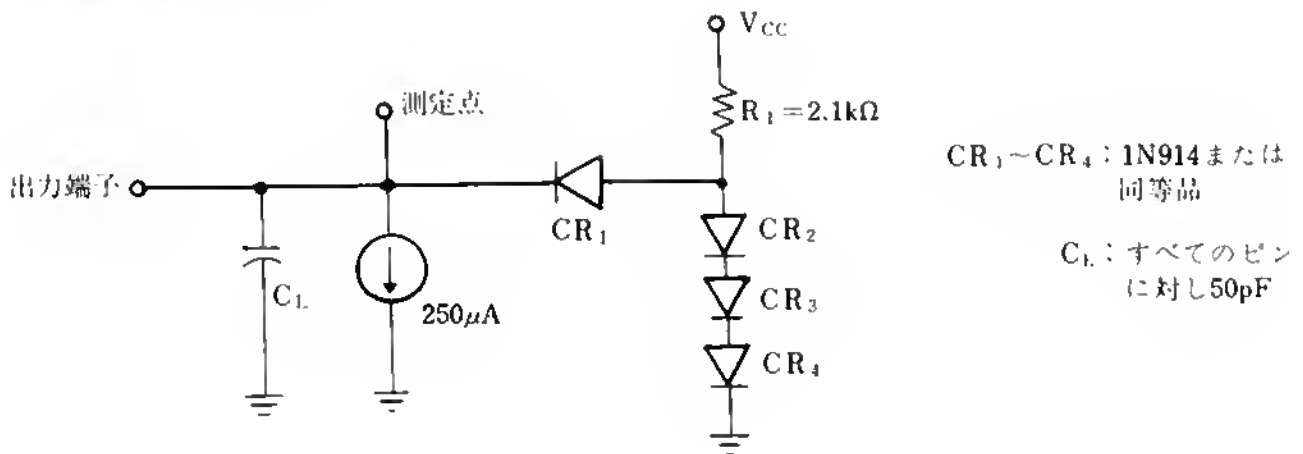
注〔1〕  $t_C = t_W(\Phi_H) + t_W(\Phi_L) + t_r + t_f$

〔2〕 負荷容量の50pF増加につき、遅延は10ns増加します。負荷容量の最大値は、データ・バスが200pFであり、他は100pFです。

〔3〕 負荷容量の10pF増加につき遅延は2ns増加します。負荷容量の最大値は100pFです。

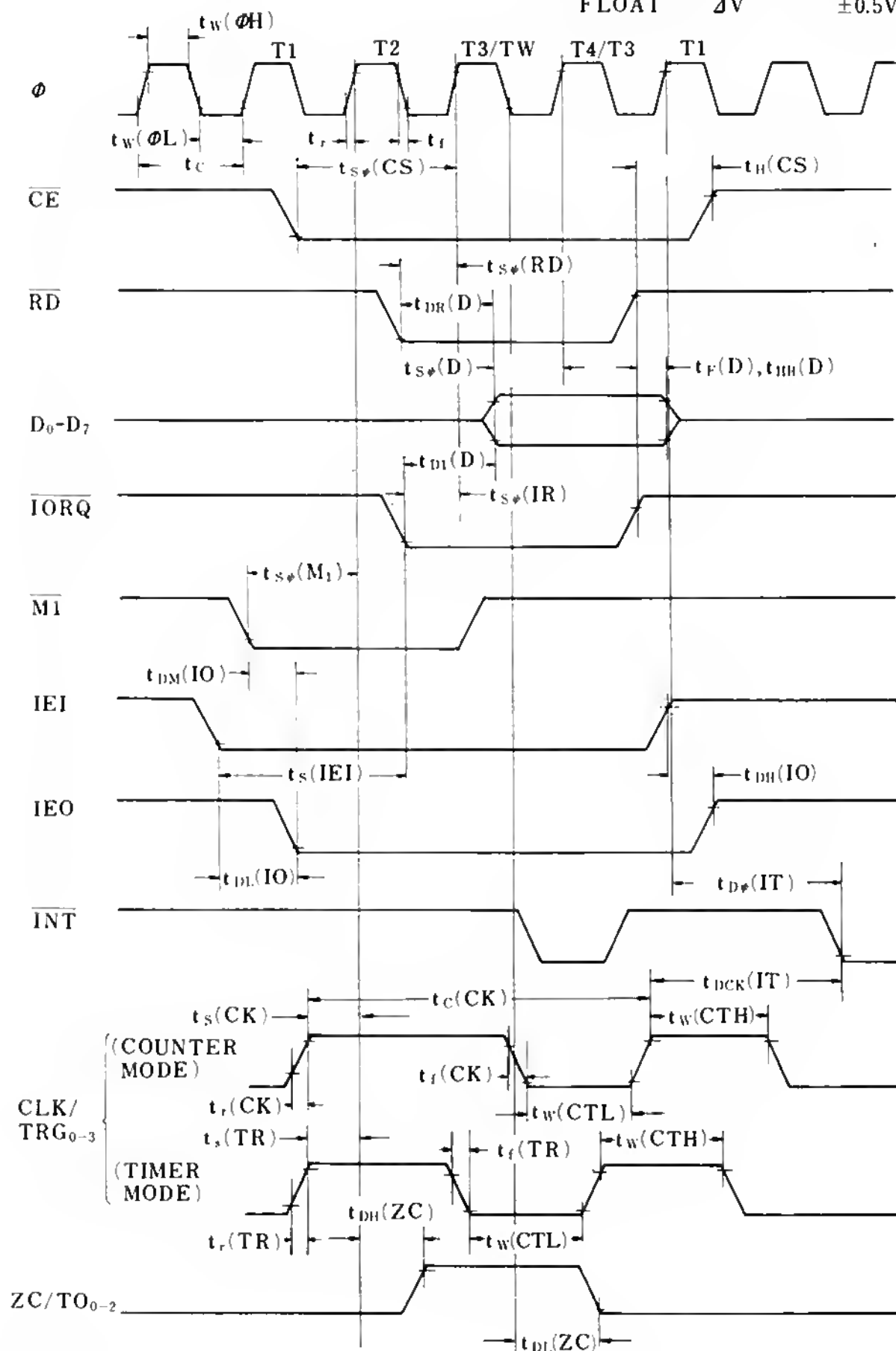
〔4〕  $\overline{\text{RESET}}$ の入力幅は最低3クロック・サイクル必要です。

### 出力端子測定回路



## AC タイミング図

	"H"	"L"
CLOCK	$V_{CC}-0.6V$	0.45V
OUTPUT	2.0V	0.8 V
INPUT	2.0V	0.8V
FLOAT	$\Delta V$	$\pm 0.5V$



## 著 者 略 歴

昭和 47 年 東京電機大学工学部  
電子工学科卒

現 在 シャープ株式会社  
電子部品営業本部

## 図解 マイクロコンピュータ Z-80 の 使 い 方

© 横田英一 1981

昭和 56 年 5 月 20 日 第 1 版第 1 刷発行  
昭和 59 年 3 月 30 日 第 1 版第 12 刷発行

OHM・OHM・OHM・O  
OHM・OHM  
著者承認  
検印省略  
O・WHO・WHO・WHO

著 者 よこ した 英 いち  
横 田 英 一

発 行 者 株式会社 オ ー ム 社  
代 表 者 種 田 則 一

発 行 所 株式会社 オ ー ム 社  
郵便番号 101  
東京都千代田区神田錦町 3-1  
振 替 東京 6-20018  
電 話 03(233)0641(大代)

Printed in Japan

組版 緑新社 印刷 秀好堂印刷 製本 大進堂  
落丁・乱丁本はお取替いたします

ISBN 4 - 274 - 07072 - 7

マイコン入門心得帖	平松・森本共著	四六判	定価	1200円
続・マイコン入門心得帖	平松・森本共著	四六判	定価	1200円
マイコン実験と工作マニュアル	北川一雄著	A5W	定価	1900円
続・マイコン実験と工作マニュアル	北川一雄著	A5W	定価	2300円
図解 初めてマイコンを学ぶ人のために だれにもわかる マイコンの考え方・使い方	大原茂之著	A5判	定価	1400円
図解 マイコンの基礎知識	大條・吹浦共著	A5判	定価	1700円
図解 マイコンの使い方	矢田光治著	A5判	定価	2300円
図解 マイコンのプログラム	小牧・大條共著	A5判	定価	1400円
図解 マイコンのためのBASIC入門	平松・守屋共著 斎藤	A5判	定価	1900円
図解 マイコンのためのアセンブラ入門	小牧・大原共著	A5判	定価	1500円
図解 マイコンのためのオペレーティングシステム入門	大原・倉田共著	A5判	定価	1500円
図解 マイコンのためのインタフェース入門	大原・倉田共著	A5判	定価	1700円
図解 マイコンのインタフェース	大原・北沢共著	A5判	定価	1700円
ミニコン・マイコン入門	平松・斎藤共著	A5判	定価	1800円
マイクロコンピュータ入門	藤井・桑原共編	A5判	定価	2000円
マイクロコンピュータの基礎	国分・藤井共著	A5判	定価	1800円
産業用マイクロコンピュータの基礎と応用	矢田光治著	A5判	定価	1500円
制御用マイコンの作り方・使い方	正田・泥堂共編 中島・松本	A5判	定価	2300円
制御用マイコンのプログラミング	北川一雄著	B5判	定価	2900円
計測自動化のためのマイコン標準インタフェース	北川一雄著	B5判	定価	3900円
マイクロコンピュータの計測と制御への応用	脇英世著	A5判	定価	2300円
エンジニアのための絵ときマイクロコンピュータ	岡田・富塚訳	B5判	定価	3400円
先生のためのマイコン教室	吉本久泰著	A5判	定価	1900円
家電技術者のためのマイコンの使い方	末武国弘監修	B5判	定価	2300円
図解 マイクロコンピュータ Z-80 の使い方	阿部・金田共著 山下	A5判	定価	2000円
図解 マイクロコンピュータ 続 Z-80 の使い方	横田英一著	A5判	定価	1700円
図解 マイクロコンピュータ Z-80 アセンブラプログラミング入門	横田英一著	B5判	定価	2800円
マイクロコンピュータ MC 6809 の考え方	湯田・伊藤共著	A5判	定価	2000円
図解 16ビットマイクロコンピュータ 8086 の使い方	曾和将容著	A5判	定価	2300円
図解 PC-8801 N88-BASIC 入門	井出裕巳著	A5判	定価	1900円
マイクロコンピュータのための PL/M 入門	佐藤達男著	A5判	定価	1900円
ワンチップマイコン入門	小牧・大條共著 佐々木・坂本 国分・浅見 野村	A5判	定価	1800円 1700円





### マイコン入門心得帖

四六判 170頁

平松啓二・森本淳堯 共著

〈主要目次〉

マイコンとは／マイコン理解のための  
基礎知識／マイコンのしくみ／マイコ  
ンを動かす／マイコンの活用／マイコ  
ンの応用と展望／マイコン用語の解説

### マイクロコンピュータの基礎

A5判 208頁

矢田光治 著

〈主要目次〉

マイクロコンピュータの概要／マイク  
ロコンピュータの基本／ハードウェア  
の実際／マイクロプロセッサの構造と  
動作／入出力機器とインタフェース／  
ソフトウェアの実際／プログラム例

### マイクロコンピュータ入門

A5判 228頁

国分明男・藤井狷介 共著

〈主要目次〉

マイクロコンピュータとは／マイク  
ロプロセッサの内部／マイクロプロセ  
ッサ、メモリ、および入出力インタフ  
ェース／マイクロコンピュータ・システ  
ム／サポートソフトウェア／ソフトウ  
ェアによる機能の拡張／マイクロコン  
ピュータの動向

### マイコン実験と工作マニュアル

A5W判 188頁

北川一雄 著

〈主要目次〉

マイコンの基礎知識／RAMを使ったプ  
ログラムタイムスイッチの試作実験／  
マシンコード式マイコンの作り方／マ  
シンコード式マイコンプログラムの研  
究／リレー式16進キーボードの作り方  
／パンチテープ式プログラム書込み器  
の作り方／バス出力ラッチ装置の作り

